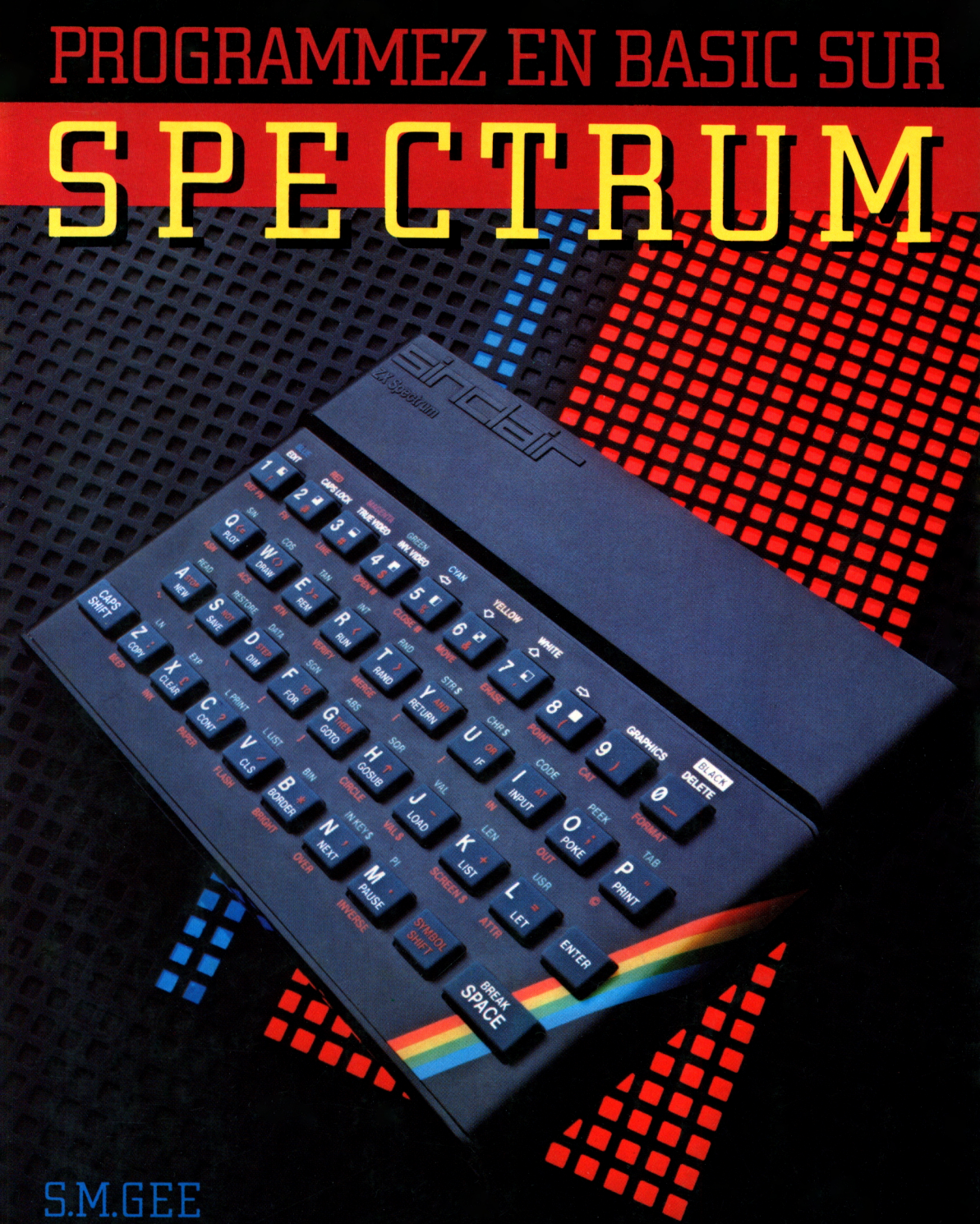
PROGRAMMEZ EN BASIC SUR

SPECTRUM



S.M.GEE

SYBEX

PROGRAMMEZ EN BASIC SUR SPECTRUM

PROGRAMMEZ EN BASIC SUR SPECTRUM

S.M.GEE

SYBEX

Paris · Berkeley · Düsseldorf

Traduction **Patrick Wajsbrot**
Photo couverture Pascal Potain

Tous les efforts ont été faits pour fournir dans ce livre une information complète et exacte. Néanmoins, SYBEX n'assume de responsabilités ni pour son utilisation, ni pour les contrefaçons de brevets ou atteintes aux droits de tierces personnes qui pourraient résulter de cette utilisation.

Copyright version originale © 1983, S.M. Gee/Granada
version française © 1983, Sybex

ISBN 2-902414-52-8

Tous droits réservés. Toute reproduction même partielle, par quelque procédé que ce soit, est interdite sans autorisation préalable. Une copie par xérographie, photographie, film, bande magnétique ou autre, constitue une contrefaçon passible des peines prévues par la loi sur la protection des droits d'auteur.

(Edition originale : ISBN 0-246-12025-8, S.M. Gee/Granada, Londres)

Table des matières

Préface, 1

1. AVANT DE VOUS CONNECTER **5**

Qu'est-ce qu'un ordinateur ?, 6
Programmes et programmation, 9
L'histoire du ZX Spectrum, 10

2. PREMIER CONTACT **13**

Mise en route du Spectrum, 15
Utilisation du clavier, 16
Entrée d'un programme, 19
Compléments sur le clavier, 22
Utilisation d'un magnétophone, 26
L'affichage et notre second programme, 29

3. VARIABLES, PRINT, LET ET INPUT **35**

Variables, 36
Mettre des choses dans une variable - LET, 37
Extraire le contenu d'une variable - PRINT, 38
Arithmétique, 39
Comprendre les expressions - L'ordre d'évaluation, 41
Variables et constantes - L'expression complète, 42
Un autre moyen de modifier les variables - INPUT, 44
Variables et constantes comme expressions, 45
Description de BASIC, 46
INPUT avec message, 47
PRINT combinés, 48

4. BOUCLES ET CHOIX - FLUX DE CONTROLE 57

Le flux de contrôle ou séquence d'exécution, 57
Boucles - L'instruction GOTO, 58
Choix et conditions - L'instruction IF, 62
Utilisation de IF, 66
L'instruction FOR, 71
Utilisation des boucles FOR, 75
IF...THEN et ":", 76

5. MANIPULATION DE TEXTES ET DE NOMBRES 83

Chaînes, 83
Expressions de chaînes, 85
Tableaux, 93
Jeu de mots, 97
Initialisation de variables - DATA et RESTORE, 98
Sauvegarde de données sur une cassette, 100

6. FONCTIONS ET SOUS-PROGRAMMES 105

Les fonctions du Spectrum, 108
Fonctions définies par l'utilisateur - DEF FN et FN, 117
Sous-programmes GOSUB et RETURN, 121
Utilisation des sous-programmes, 123

7. LES GRAPHIQUES 125

Contrôler l'instruction PRINT, 125
Fonctions d'affichage - TAB et AT, 128
Ecran plein - CLS et le défilement, 131
Contrôler INPUT, 132
Les caractères graphiques, 135
Caractères graphiques définis par l'utilisateur, 139

Modifier l'affichage - INVERSE et OVER, 142
Attributs de caractères - FLASH et BRIGHT, 145
Couleur - BORDER, INK et PAPER, 146
Commandes d'affichage en couleur, 149
Couleurs temporaires, 150
Utilisation des graphiques dans les jeux, 151

8. LE SON ET LES JEUX **153**

Les sons simples - BEEP, 153
Programmation des airs de musique, 156
Silence - PAUSE, 160
Quelques effets sonores, 161
SCREEN$, la fonction d'écran, 163

9. GRAPHIQUES HAUTE RESOLUTION **169**

L'écran haute résolution, 169
Les commandes graphiques - PLOT, DRAW et CIRCLE, 171
Couleurs haute résolution, 177
Inverse de PLOT - OVER et INVERSE, 179
Découvrir ce qui est sur l'écran - POINT, 180
Utilisation des graphiques haute résolution, 181

10. LOGIQUE ET AUTRES SUJETS **185**

Logique et expressions conditionnelles, 186
BIN, PEEK, POKE, IN, OUT, USR et CLEAR, 189
Extraire une couleur - ATTR, 193
REM (REMarques), 194

Conclusion, 195

Index, 196

PREFACE

Ce livre a été écrit à l'intention de tous les utilisateurs de Spectrum désireux d'apprendre à écrire leurs propres programmes. Programmer votre Spectrum peut être amusant et vous apporter beaucoup de satisfactions ; en outre, l'étude du BASIC n'est en aucune façon une entreprise effrayante. Ce livre a pour but de vous apprendre à tirer le meilleur parti de votre microordinateur d'une façon progressive et claire.

Le Chapitre 1 expose quelques idées générales concernant les ordinateurs et la programmation ainsi que l'arbre généalogique du Spectrum. Le Chapitre 2 vous prépare à apprendre la programmation sur votre machine et propose un programme vous permettant d'accorder le téléviseur avec le Spectrum grâce à sa carte test. Nous commencerons à apprendre la programmation au Chapitre 3 qui présente les variables et quelques-uns des mots clés tels que : PRINT, INPUT et LET. Le Chapitre 4 introduit la notion de séquence

d'instructions sur laquelle repose la programmation des ordinateurs. Le Chapitre 5 montre comment votre Spectrum traite les nombres et les mots, et la façon dont vous pouvez les combiner dans vos programmes. L'utilisation des fonctions du Spectrum ainsi que la notion de sous-programme sont les thèmes principaux du Chapitre 6, et c'est ici que les instructions très importantes RND, RANDOMISE et INKEY$ seront étudiées. Les graphiques et l'utilisation de la couleur sont traités en détail au Chapitre 7. Le son est abordé au Chapitre 8 qui présente un programme de jeu utilisant des graphiques en couleur et des effets sonores. Le Chapitre 10 explique les possibilités de graphisme haute résolution et présente quelques programmes pour les illustrer. Le Chapitre 9 enfin traite les expressions logiques et introduit quelques ordres permettant au programmeur d'accéder au coeur même du Spectrum.

Comme vous le voyez, ce livre développe un grand nombre de sujets. Cependant il a été conçu de manière à rester avant tout divertissant. Ce n'est pas un manuel de théories mais surtout un livre d'exercices pratiques. Vous y trouverez beaucoup de programmes et éléments de programmes. En conclusion, appréciez votre Spectrum et amusez-vous en apprenant le BASIC.

Nous avons eu beaucoup de plaisir à écrire ce livre qui nous a permis de découvrir des choses intéressantes sur le Spectrum que nous pensons être un remarquable micro-ordinateur ainsi que sur le BASIC qui est certainement le meilleur langage de programmation à ce jour.

1

AVANT DE VOUS CONNECTER

Lorsque vous regardez un Spectrum pour la première fois, vous êtes étonné de sa petite taille. Vous devriez l'être plus encore si vous saviez qu'il y a à peine vingt ans un ordinateur de même capacité mémoire aurait rempli toute une maison. Vous devez vous demander si une machine aussi petite peut réellement faire toutes les choses dont vous avez entendu parler. Et bien elle peut le faire ! Elle peut faire des calculs compliqués, afficher des graphiques en couleur et émettre différents sons et bien d'autres choses encore. En outre, sa seule limite n'est pas conditionnée par sa petite taille mais par son coût. Cependant, le fait que la micro-technologie moderne soit responsable de cette réduction de la taille des machines est une chose que l'utilisateur de Spectrum peut parfaitement ignorer ; ce qui est important, c'est que l'ordinateur soit accessible à tous ceux qui le désirent.

QU'EST-CE QU'UN ORDINATEUR ?

Cette question peut être traitée à plusieurs niveaux. Des chapitres entiers, voire même des livres complets peuvent être consacrés à ce sujet. Pour vous servir de votre Spectrum, vous n'avez pas besoin de vous poser cette question. Après tout, nous regardons tous la télévision, sans que nous nous posions la question : qu'est-ce exactement que la télévision ? Si vous voulez connaître le moindre détail de fonctionnement du Spectrum, il n'y aura alors pas d'autre solution que d'apprendre l'électronique. Cependant il n'est pas difficile de comprendre ce que fait un ordinateur, et approximativement comment il le fait, sans rien connaître de l'électronique ni de l'omniprésente puce ! L'important est de savoir qu'aucun ordinateur n'aurait pu exister si l'électronique n'avait pas été inventée. En vérité, le premier ordinateur qui fut construit utilisait des pignons et des engrenages, et ce, cent ans avant qu'une diode (composant électronique récent) ne trouve sa place dans une machine. Bien qu'en pratique l'ordinateur semble être un produit de la technologie des microprocesseurs, le concept même de l'ordinateur ne dépend pas du choix des matériaux avec lesquels il est construit.

Chaque ordinateur est composé d'un certain nombre d'éléments ayant chacun des fonctions bien déterminées. Ainsi, tout ordinateur doit avoir un moyen de communication avec le monde extérieur. Dans le cas de votre Spectrum, ceci est réalisé par le clavier sur lequel vous tapez, et l'écran de télévision que l'ordinateur utilise pour vous montrer ce que vous lui avez dit ou toute autre chose qu'il doit vous transmettre. Le clavier est un exemple d'unité d'entrée (Input) et la télévision est une unité de sortie (Output). Mais ceci ne représente pas les seules unités d'entrée-sortie (I/O, Input-Output) qui peuvent être utilisées par un ordinateur. Vous pouvez acheter, par exemple, une petite imprimante compatible avec le Spectrum, et ainsi commander à l'ordinateur d'afficher vos sorties aussi bien sur papier que sur l'écran.

Une machine qui ne serait capable que de recevoir des informations et de les transmettre sans les avoir traitées, ne pourrait pas s'appeler un ordinateur. Elle serait plutôt à classer parmi les téléphones ou les télex ! A l'intérieur d'un ordinateur, il y a un mécanisme qui permet de changer ou traiter l'information avant de la restituer. Ce mécanisme est habituellement réalisé par une électronique compliquée, cachée dans l'ordinateur. Ce dont nous venons de parler est connu sous le nom d'Unité Centrale ou UC (Central Processing Unit, CPU).

Dans le Spectrum l'UC est contenue dans une simple puce appelée Z80 et c'est là bien sûr l'origine du "Z" dans son nom complet : le ZX Spectrum. Est-il important pour programmer en BASIC de connaître ce que fait le Z80 et comment il le fait ? Certainement pas ! Cependant, il faut savoir que le Z80 exécute des opérations arithmétiques et logiques sur les informations entrées à partir du clavier et stockées dans la machine. Ces opérations sont contrôlées par une liste d'instructions appelée programme. Cet aspect de l'ordinateur est si important que vous pourriez presque dire que c'est une machine qui obéit à une liste d'instructions. Mais toute espèce de définition d'une machine aussi complexe qu'un ordinateur est dangereuse ! Les différents types d'instructions auxquelles obéit le Spectrum seront le thème développé dans tout le reste de ce livre, aussi pour le moment les laisserons-nous de côté.

Si un ordinateur peut traiter les informations grâce à une liste d'instructions, il est bien évident qu'il doit avoir stocké quelque part, non seulement ces informations, mais également cette liste d'instructions. Cette partie de la machine est connue sous le nom de mémoire, mais le terme RAM (Random Access Memory, mémoire à accès aléatoire) est plus couramment utilisé. Vous pouvez imaginer qu'une RAM est une sorte de bloc-notes sur lequel l'UC peut enregistrer la liste d'instructions ainsi que toutes les données dont elle a besoin. Evidemment, une mémoire a des limites, et sa capacité est un critère commode pour déterminer la puissance d'un ordinateur. Plus la mémoire est importante, plus la

liste d'instructions à stocker peut être grande. L'unité de mémoire de l'ordinateur la plus commode est l'octet (byte). Une mémoire d'un octet peut contenir un caractère. (Ici, le mot caractère signifie une lettre, un chiffre ou n'importe quel signe de ponctuation que vous pourriez rencontrer dans un texte tel que ce livre !) Ainsi une mémoire de 400 octets serait suffisante pour contenir le quart d'une page de ce livre. Le seul inconvénient de cette unité de mesure est d'être un peu trop petite. Les ordinateurs disposant généralement de mémoires permettant de stocker des milliers de caractères, il est plus commode d'utiliser un multiple : le Kilo Octet, souvent abrégé en KO ou encore plus simplement K. Pour différentes raisons, un KO ne représente pas 1000 octets comme on pourrait le supposer, mais 1024 octets. (Vous pouvez remarquer que ce nombre étrange représente la puissance de deux la plus proche de 1000 et ce, comme vous le savez déjà, parce que les ordinateurs travaillent en binaire, système basé sur deux états.) Il existe deux versions de Spectrum qui ne diffèrent que par leur capacité mémoire : 16K ou 48K. Ainsi, le plus petit Spectrum peut stocker environ 16000 caractères, ce qui est suffisant pour bon nombre d'applications intéressantes. (Auparavant, les ordinateurs utilisés par l'armée, pour calculer la trajectoire des missiles, etc., disposaient souvent de moins de 16K.)

Cette association d'unités d'entrée-sortie, UC et mémoire, définit un ordinateur. Les entrées-sorties communiquent avec le monde extérieur, l'UC calcule et traite les informations, et la mémoire retient la liste des instructions à exécuter ainsi que les données à traiter par l'UC. En pratique, il y a cependant une chose à ajouter à cette liste. Lorsque vous éteignez votre Spectrum, il perd tout le contenu de sa mémoire. Pour conserver les informations, la mémoire des ordinateurs a un besoin permanent d'électricité. Si vous coupez l'alimentation, vous perdez vos informations. Cette sorte de mémoire est connue sous le nom de mémoire volatile. Cette perte de mémoire est problématique car elle oblige à taper la liste d'instructions chaque fois que l'appareil a

été éteint. Pour éviter cela, beaucoup d'ordinateurs disposent d'un second type de mémoire non volatile. En ce qui concerne le Spectrum, cette mémoire est obtenue à l'aide d'un simple magnétophone à cassettes, qui permet de sauvegarder les programmes et les données tels qu'ils figuraient en mémoire avant de couper le courant. Un deuxième avantage de ce type de mémoire est qu'elle est transportable. Vous pouvez enregistrer un programme sur une cassette puis l'enlever du magnétophone (et souvent la prêter à quelqu'un d'autre). Vous pouvez alors, soit repartir sur un nouveau programme, soit revenir sur un ancien en le chargeant à partir d'une bande précédemment enregistrée. Le Spectrum peut utiliser un deuxième type moins courant de stockage amovible : le microdrive. Son fonctionnement est similaire au magnétophone à ceci près qu'il est plus rapide. Il peut être acheté comme complément au Spectrum. Si vous êtes expérimenté ou si vous le devenez d'ici à la fin de ce livre, un microdrive vous sera très utile.

PROGRAMMES ET PROGRAMMATION

Comme il a été dit précédemment, un ordinateur obéit à une liste d'instructions stockées dans sa mémoire. Cette liste d'instructions est appelée un programme et sa rédaction, la programmation. On croit souvent que la programmation est une activité qui fut créée avec les ordinateurs modernes, alors qu'en fait, des individus conçurent des règles (comparables à des listes d'instructions) auxquelles d'autres personnes obéissaient avant même que l'écriture ne fut inventée. Vue sous cet angle, la programmation ne représente rien de nouveau. Un des meilleurs exemples de programmation traditionnelle est peut-être l'écriture de la musique. Vous pouvez comparer une partition musicale à un programme qui indiquerait à un musicien comment jouer un air particulier. De fait l'écriture de la musique ressemble à un programme d'ordinateur en ce sens qu'elle nécessite l'utilisation d'un

langage beaucoup plus précis que le langage habituel. Une seule note déplacée et vous obtenez un air différent ! De même un programme d'ordinateur s'écrit dans un langage particulier également très précis. Dans le cas du Spectrum, ce langage est le BASIC, le plus populaire des langages de programmation. De même que dans une partition musicale, une modification dans un programme BASIC peut en changer complètement la signification. Aussi devez-vous prendre conscience qu'il faut être très attentif en apprenant le BASIC, et ce, dès le début. A la différence d'une langue, où vous pouvez commencer par apprendre des mots, des phrases, et ensuite la ponctuation, vous devez connaître la signification de la moindre virgule d'une ligne BASIC pour qu'elle ait un sens !

Par contre, à l'inverse du français, vous trouverez rarement des variantes d'orthographe ou de règles de ponctuation en BASIC. De plus le BASIC s'appuie sur quelques idées maîtresses qui, dès lors que vous les connaîtrez, vous aideront à comprendre le pourquoi des règles. Au fur et à mesure que vous avancerez dans ce livre, vous apprendrez deux choses : les détails les plus précis de chacun des ordres BASIC d'une part et les caractéristiques générales communes à tous les langages de programmation d'autre part.

L'HISTOIRE DU ZX SPECTRUM

Avant de parler du Spectrum, il peut être intéressant de jeter un bref regard sur son arbre généalogique. En 1980, Sinclair Research lança un petit ordinateur, le ZX80, mettant ainsi l'informatique à la portée de presque tous. Il était cependant très limité. C'était une révolution mais, pour plusieurs raisons, il arrivait trop tôt. Il pouvait exécuter de petits programmes BASIC, mais les seuls traitements arithmétiques qu'il pouvait effectuer se limitaient aux nombres entiers. Il ne pouvait pas afficher des informations sur l'écran de télévision et traiter simultanément des données. Dès qu'il faisait quelque chose

l'écran tremblotait. C'est pourquoi un grand nombre de gens qui achetèrent le ZX80 pour des raisons diverses furent déçus de constater qu'il ne répondait pas à leurs espérances.

En 1981, Sinclair lança le successeur du ZX80, le ZX81. Le ZX81 est en fait le premier ordinateur réellement utilisable digne de figurer dans ce bref historique. Il disposait d'une très petite mémoire (1K), mais néanmoins, l'écran ne tremblotait plus et pour la première fois, il était possible d'obtenir de bons graphiques, statiques ou animés. Avec le ZX81, apparut le ZX-BASIC qui est utilisé dans une version étendue sur le Spectrum. Le ZX-BASIC est une version très perfectionnée du BASIC qui offre beaucoup d'avantages par rapport à celui d'autres machines. Pour remédier à la petite taille de la mémoire, Sinclair élabora une mémoire additionnelle de 16K RAM sous forme de module et pour étendre les possibilités de la machine, une petite imprimante de coût modique fut diffusée. Le module de 16K RAM ne peut être utilisé qu'avec le ZX81 ; en revanche, l'imprimante reste compatible avec le Spectrum.

En 1982, Sinclair annonçait le Spectrum en complément du ZX81. Le Spectrum ajoute le son, la couleur et le graphisme haute résolution aux possibilités du ZX81, et en général des performances accrues pour une faible augmentation de prix. De plus, Sinclair propose une unité de stockage à très bas prix, le "microdrive", ainsi qu'une interface de communications (aucun de ces deux éléments ne peut être utilisé avec le ZX81). Ces deux derniers éléments assurent au Spectrum pour les années à venir une bonne place sur le marché, autant pour des applications professionnelles que personnelles.

Le Spectrum est une machine logique et performante, construite en s'appuyant sur l'expérience acquise avec les ZX80 et ZX81. De ce fait, il devrait bénéficier d'une grande diffusion. Le ZX-BASIC, pour sa part, a lui aussi beaucoup progressé et occupe une place de choix parmi les langages de programmation. En conséquence, c'est une très bonne option pour le présent et pour l'avenir que d'apprendre ce BASIC.

2

PREMIER CONTACT

En utilisant un ordinateur, on rencontre deux types de problèmes. Le premier consiste simplement à savoir le brancher et connaître ses commandes. Le second est la rédaction de programmes opérationnels. Ce chapitre se propose de traiter ces deux problèmes, avant de passer à notre tâche principale qui est l'apprentissage du BASIC.

UNE NOUVELLE CONNAISSANCE

Apprendre à connaître un ordinateur est un problème, même si vous êtes chevronné. Bien que beaucoup de points soient communs entre différentes machines, il existe toujours suffisamment de petites différences, pour que le passage d'un ordinateur à un autre nécessite une période d'adaptation. Par exemple, presque tous les ordinateurs

utilisent un clavier standard de machine à écrire mais cependant, avec de nombreuses touches supplémentaires dont l'emplacement varie d'une machine à l'autre, ce qui ne manque pas au début de désappointer même les plus avertis. Bien sûr, si vous êtes expérimenté, vous savez que cette période d'adaptation est rapide, mais si vous êtes débutant, vous pouvez vous décourager et penser que la manipulation d'un ordinateur est une chose insensée ! L'ennui est que, si vous ne vous sentez pas à l'aise avec votre ordinateur, vous aurez tendance à penser que la programmation n'est pas une chose facile.

Il n'y a aucun moyen d'éviter cette première barrière de la programmation, car de façon imagée, avoir de bons rapports avec son ordinateur n'est qu'une question de temps et de pratique. Vous supporterez plus facilement cette période frustrante, si vous acceptez les conseils suivants :

1. Séparez dans votre esprit les problèmes liés à l'utilisation de votre Spectrum des difficultés que vous pourriez avoir avec la programmation.

2. N'imputez pas immédiatement à votre machine toute réaction surprenante de sa part. A priori, il y aura toutes les chances pour que l'erreur vienne de vous !

3. Ne croyez pas d'emblée que le comportement inattendu d'un programme signifie que le Spectrum est illogique. Les ordinateurs ont une logique impitoyable. Essayez de ne pas confondre les erreurs de frappe avec les erreurs de programmation.

Pour vous aider à cerner cette difficulté initiale, et à la surmonter, ce chapitre vous propose deux petits programmes que vous devriez tenter de faire tourner avant de poursuivre la lecture de ce livre. Ce sont des programmes complets et opérationnels, qui vous sont présentés pour mettre en évidence les problèmes d'utilisation de la machine n'ayant rien à voir avec la programmation.

MISE EN ROUTE DU SPECTRUM

Le Spectrum est un des ordinateurs les plus simples à mettre en marche. Tout ce que vous avez à faire est de brancher le bloc d'alimentation sur une prise de courant (la petite boîte noire marquée ZX POWER SUPPLY) puis d'enfoncer le petit jack cylindrique dans la prise située derrière l'appareil, marquée 9V DC. A ce moment-là vous devez entendre un son aigu comparable à celui d'un moustique, en provenance de la machine. Si vous n'entendez rien, vérifiez que l'alimentation est bien connectée. Si malgré cela vous ne percevez toujours rien, appuyez sur la touche A, à gauche, sur le clavier. En maintenant cette touche enfoncée, vous devez entendre un cliquetis. Si ce test échoue, votre machine est en panne.

Brancher le Spectrum est la seule opération à effectuer pour le mettre en service. Cependant, pour voir ce qu'il réalise, il vous faut le raccorder à un récepteur de télévision UHF en état de marche ! Cette opération est un peu plus compliquée par le fait qu'elle requiert un appareil qui varie d'une maison à l'autre, le téléviseur. Une fois l'ordinateur raccordé au téléviseur, il vous fera penser à une nouvelle chaîne de télévision. Eh oui ! le Spectrum n'est pas seulement un ordinateur, c'est aussi une petite station de télévision. Quand votre récepteur vous a été livré la première fois, il a été réglé pour recevoir les émetteurs de votre région. De la même façon, lorsque vous utilisez votre Spectrum pour la première fois, vous devez régler votre téléviseur pour qu'il le reçoive. Si votre récepteur est pourvu de boutons poussoirs pour la sélection, alors vous pourrez choisir quel sera celui de la chaîne Spectrum et régler votre poste en vous référant à sa notice d'utilisation ; s'il est muni d'un cadran d'accord, la chaîne Spectrum se règle de la même manière que n'importe quelle autre chaîne.

Lorsque vous arriverez sur la chaîne Spectrum, vous commencerez à apercevoir un embryon d'image. Continuez

jusqu'à ce que vous puissiez voir le message :

©1982 Sinclair Research Ltd.

clairement dans le bas de l'écran. Comme pour n'importe quelle autre chaîne, si vous ne l'avez pas accordée exactement, la qualité de l'image sera médiocre, aussi soyez patient et faites attention à la précision du réglage jusqu'à ce que vous obteniez une belle image.

Si vous disposez d'un récepteur couleur, vous devez régler la brillance, le contraste et les couleurs pour obtenir le meilleur résultat possible. Mais ne faites ces réglages qu'après avoir obtenu l'image la plus nette possible. Si vous utilisez un poste noir et blanc, vous ne pourrez pas voir les couleurs générées par le Spectrum, mais vous devez tout de même régler la brillance et le contraste pour obtenir une qualité d'image satisfaisante. Un récepteur noir et blanc fonctionnera parfaitement avec le Spectrum, mais au lieu de voir les différentes couleurs, vous obtiendrez un dégradé de huit tons de gris allant du noir au blanc.

Si vous le souhaitez, vous pourrez effectuer les réglages définitifs après avoir entré le programme donné plus loin dans ce chapitre, car il utilise la carte de test du Spectrum qui pourra vous aider à faire la mise au point.

UTILISATION DU CLAVIER

Après avoir appris la mise en route de votre Spectrum, l'opération suivante consiste à apprendre à vous servir du clavier. La particularité la plus frappante du Spectrum est peut-être le nombre de lettres et de mots écrits sur les touches. L'idée d'avoir plus d'une fonction par touche du clavier n'est pas nouvelle. Les machines à écrire utilisent une touche unique pour imprimer les minuscules et les majuscules, sans que personne ne s'étonne du fait que la sélection entre les deux se fasse au moyen d'une touche particulière : la touche d'échange (shift key). Remarquez

que la touche Shift ne produit rien du tout si vous l'actionnez seule, ainsi est-elle différente des autres touches du clavier.Comme elle contrôle ce que produisent les autres, elle est appelée touche de contrôle.

Le Spectrum dispose d'un certain nombre de touches qui, combinées, permettent d'accéder à tous ses caractères d'exécution ou d'impression. La Figure 2.1 montre leur emplacement sur le clavier. La fonction de chacune de ces touches sera expliquée dans ce chapitre, en commençant par celles dont on se sert le plus souvent.

Le Spectrum utilise deux touches permettant de choisir chacun des trois caractères figurant sur chaque touche. La touche Caps Shift agit de la même façon que la touche majuscule sur une machine à écrire. Si vous appuyez isolément sur une touche, vous obtiendrez la version minuscule de la lettre correspondante. A l'inverse, si vous

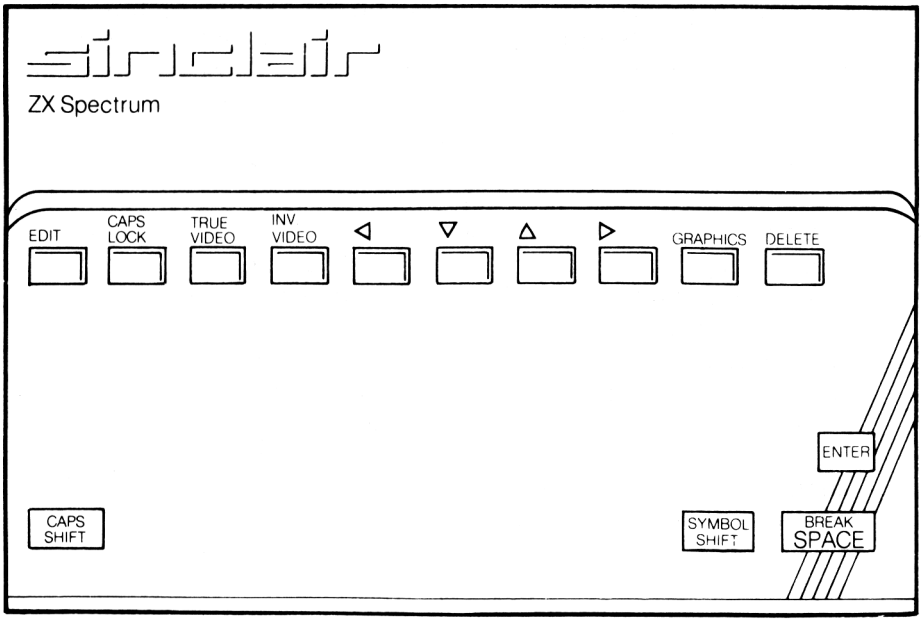


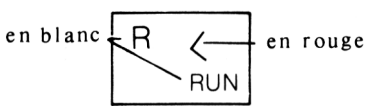
Figure 2.1 : Position des touches spéciales

pressez une touche en même temps que Caps Shift, vous obtiendrez la lettre majuscule.

La deuxième touche spéciale est Symbol Shift située à l'extrême droite du clavier. Actionner n'importe quelle touche du clavier en même temps que celle-ci, produit le mot ou le symbole noté en rouge, à droite de la lettre ou du chiffre principal.

L'utilisation des deux touches Shift semble assez facile ; cependant si vous venez juste de brancher le Spectrum et que vous actionnez la touche A, vous serez surpris de voir apparaître sur l'écran le mot NEW. Quelque chose a été mal expliqué ! D'après ce qui précède, la touche A aurait dû produire "a". La raison de cette anomalie est que chaque ligne BASIC débute avec l'un de ces petits mots, les mots clés. Dans un but de simplification, le Spectrum interprète la toute première lettre que vous frappez, comme un mot clé. Ce mot clé généré par chaque touche est affiché en blanc en bas de l'écran. Le mot clé correspondant à A est NEW ; c'est la raison pour laquelle ce mot apparaît à l'écran lorsque vous actionnez cette touche immédiatement après la mise en route. Si vous tapez A une seconde fois, la lettre "a" apparaît alors comme prévu. Afin d'indiquer ce que va produire la touche que vous êtes en train d'actionner, le Spectrum affiche un caractère spécial. Celui-ci est appelé le curseur, sa position sur l'écran indique l'emplacement du prochain caractère à afficher et la lettre qu'il contient indique le mode dans lequel il se trouve placé. Au début de chaque ligne, le curseur est représenté par K clignotant jusqu'à ce qu'un mot clé soit entré, à la suite de quoi il se transforme en L clignotant (K pour Keyword, mot clé et L pour Letter, lettre). Dès que le curseur devient L clignotant, les deux touches Shift agissent comme il a été expliqué précédemment.

Pour résumer, examinons la touche suivante :



- Les mots clés en blanc dans le bas de chaque touche sont produits lorsque le curseur est symbolisé par K clignotant. Dans le cas de notre touche exemple, le mot RUN serait affiché.

- Les majuscules et les minuscules sont obtenues lorsque le curseur est L clignotant, minuscules quand Caps Shift n'est pas enfoncé ("r" dans notre exemple) et majuscules sinon ("R" dans l'exemple).

- Les symboles en rouge, imprimés dans le coin en haut et à droite de chaque touche, sont produits en appuyant simultanément sur la touche portant le symbole et la touche Symbol Shift, < (symbole inférieur à) dans notre exemple. Ce symbole est produit aussi bien avec le curseur K que L.

Nous savons maintenant comment obtenir quatre des symboles ou mots figurant sur chaque touche, et ceci est suffisant pour entrer un petit programme. Il y a d'autres fonctions prévues sur le clavier mais nous y reviendrons plus tard.

ENTREE D'UN PROGRAMME

Comme il a été dit au Chapitre 1, un programme est une liste d'instructions à laquelle la machine obéit. En BASIC cette liste est construite en tapant des lignes contenant des commandes, chaque ligne débutant par un numéro - le numéro de ligne. Le meilleur moyen de comprendre cela est d'essayer un petit programme. Ne vous effrayez pas si vous ne comprenez pas comment il s'exécute, concentrez-vous uniquement sur le fait de l'entrer correctement. Avant de commencer, éteignez votre Spectrum, puis après quelques secondes remettez-le en marche. Ceci aura pour effet de réafficher le message de Copyright, maintenant connu, et surtout d'être certain que tout ce que vous aviez entré pendant votre prise de contact, a bien disparu de la machine. Maintenant tapez la ligne suivante en prenant garde

de ne pas faire de faute. (Si vous faites une faute, éteignez et recommencez. Ceci est un moyen provisoire d'annuler vos erreurs, d'autres méthodes plus satisfaisantes seront exposées plus tard.)

```
10 LET a=0
```

Remarquez que le mot LET est un mot clé obtenu en tapant L lorsque le curseur est sous la forme K clignotant. Notez que le deuxième et le dernier caractère sont des zéros, et non pas la lettre O (le zéro est situé à l'extrême droite de la première rangée des touches). Le signe = est en rouge sur la touche L et doit donc être obtenu en pressant simultanément Symbol Shift et L.

Après avoir tapé cette ligne, vous devez appuyer sur la touche marquée ENTER (entrée) à l'extrême droite du clavier. Le but de cette dernière commande est de dire au Spectrum que vous avez fini de taper la ligne, et qu'il peut donc essayer de l'insérer dans un programme qui a pu être tapé précédemment. Essayez, car bien que vous pensiez avoir tapé correctement, le Spectrum effectue des contrôles, et s'il trouve des erreurs, il rejette la ligne. Lorsque l'ordinateur accepte la ligne, celle-ci disparaît du bas de l'écran pour se réafficher en haut. A présent, entrez la ligne suivante :

```
20 PRINT a
```

Encore une fois, le mot PRINT est un mot clé qui est entré en appuyant sur la touche P. Pressez ENTER, et la seconde ligne apparaîtra à nouveau dans la partie supérieure de l'écran juste après la première ligne. Entrez ensuite la ligne suivante :

```
30 LET a=a+1
```

Cette fois, avant de presser ENTER, regardons comment vous pourriez corriger d'éventuelles erreurs. Supposez par

exemple que vous ayez tapé "b=a+1" par inadvertance ; vous seriez heureux alors d'apprendre que le Spectrum vous offre un retour arrière (back space), qui vous permet de modifier très facilement ce que vous avez tapé. Si vous pressez la touche représentant une flèche à gauche (la cinquième sur la rangée du haut) en même temps que Caps Shift, vous verrez le curseur se déplacer par-dessus les lettres de la ligne. Si vous appuyez sur la touche DELETE, (le 0 sur la rangée du haut), en même temps que Caps Shift, la lettre ou même le mot clé situé à gauche du curseur disparaîtra (il sera supprimé). Vous pourrez alors insérer un nouveau mot clé ou une lettre en le tapant simplement à cet endroit. De même vous pouvez déplacer le curseur vers la droite à l'aide de la touche flèche à droite (la huitième en haut) et quel que soit le caractère que vous taperez, il sera inséré à gauche du curseur. (Ceci est plus facile à exécuter qu'à expliquer, aussi n'ayez pas peur de vous entraîner... Vous ne risquez pas d'abîmer votre Spectrum.) Finalement, quand vous avez terminé votre ligne, pressez ENTER puis tapez :

```
40 GOTO 20
```

Remarquez que lorsque vous enfoncez la touche marquée GOTO, les deux mots GO TO (aller à) apparaissent sur l'écran.

Vous devriez avoir maintenant le programme suivant affiché en partie haute de l'écran :

```
10 LET a=0
20 PRINT a
30 LET a=a+1
40 GOTO 20
```

Ceci est une suite d'instructions à laquelle vous pouvez soumettre le Spectrum en entrant le mot clé RUN puis en appuyant sur ENTER. Mais avant de l'essayer, il est plus important de remarquer qu'il y a deux façons de faire réagir le Spectrum aux ordres. Si vous tapez un ordre sans numéro de ligne, puis ENTER, le Spectrum exécutera cet ordre seul :

ceci est appelé le mode immédiat. Alors que, si vous le faites précéder d'un numéro de ligne, puis de ENTER, cet ordre sera ajouté au programme se trouvant en mémoire : ceci est appelé le mode différé. Lorsque vous tapez RUN, il n'y a pas de numéro de ligne, aussi l'ordinateur exécute cet ordre immédiatement.

Lorsque vous taperez RUN, vous verrez s'afficher sur l'écran les nombres de 0 à 21 puis une question "Scroll ? y/n" (défilement ? oui/non). Si vous appuyez sur la touche Y, vous découvrirez que l'écran est remonté d'une ligne afin de faire de la place pour la suite. Après que ceci se soit produit 22 fois, la question sera à nouveau posée. La raison pour laquelle cela se passe ainsi, est que le Spectrum essaie de vous montrer le maximum de ce que contient l'écran avant de l'effacer.

Vous pouvez ainsi continuer à répondre y (pour oui) à "Scroll ?" jusqu'à ce que vous en ayez assez de voir défiler des nombres. Vous répondez alors n (non) à la question suivante, puis pressez ENTER. Vous reverrez alors apparaître la liste du programme. Si vous désirez interrompre le programme à n'importe quel moment, appuyez sur la touche BREAK en même temps que Caps Shift. Comme autre exemple d'ordre en mode immédiat, tapez LIST (la touche marquée K) puis RETURN. Ceci provoquera l'affichage du programme que vous avez tapé précédemment.

COMPLEMENTS SUR LE CLAVIER

Il n'est jamais très compliqué d'entrer un programme, cependant il subsiste encore quelques mots et symboles dont nous avons besoin de savoir comment les obtenir. En particulier, nous ne savons pas encore comment produire des mots ou symboles tels que ceux figurant sur le boîtier du Spectrum. En fait tous ces symboles et mots sont entrés avec le curseur en un mode différent. De la même manière que les mots clés sont produits lorsque le curseur est un K, les symboles en rouge et en vert sont entrés en mode appelé mode

étendu avec le curseur représentant un E. Vous pouvez mettre le Spectrum en mode étendu en pressant simultanément les deux touches Shift. Les mots en vert au-dessus de chaque touche sont produits simplement sans l'aide des touches supplémentaires (le curseur étant en mode étendu bien sûr), alors que les mots en rouge sont obtenus en pressant en même temps la touche Symbol Shift. Il est facile de retenir que les symboles ou mots en rouge sont obtenus à l'aide de Symbol Shift, en remarquant que la touche est elle-même écrite en rouge. Ainsi pour entrer RND (au-dessus de la touche T) vous devez d'abord entrer en mode étendu avec Caps Shift et Symbol Shift et ensuite appuyer sur T. Si vous voulez activer BEEP (en-dessous de la touche Z), entrez d'abord en mode étendu puis appuyez sur Symbol Shift et Z. Vous pouvez sortir du mode étendu dès que vous tapez n'importe quoi, cependant si vous voulez l'abandonner sans rien taper, appuyez à nouveau sur les deux touches Shift.

Pour résumer ce que nous venons d'apprendre, reprenons l'exemple précédent. La figure suivante montre la touche R et ses fonctions environnantes.

- Les mots en vert au-dessus de chaque touche sont obtenus avec le curseur E après que les deux touches Shift aient été pressées. Une fois le E affiché, INT sera obtenu en tapant sur la touche R.

- Les mots en rouge au-dessous de chaque touche sont obtenus avec le curseur E et une action simultanée sur l'une des touches Shift, puis la touche correspondante (R dans notre exemple qui provoquera l'affichage de VERIFY).

Maintenant vous savez comment obtenir presque tous les symboles et mots du clavier. Les seuls qui puissent encore

vous troubler sont les mots écrits en blanc au-dessus de la première rangée de touches. Ceux-ci doivent être entrés dans un autre mode, le mode graphique. Bien que le sujet des graphiques soit traité entièrement dans le Chapitre 7, nous indiquerons ici comment y accéder. Pour entrer en mode graphique, tout ce que vous devez faire est d'appuyer simultanément sur Caps Shift et Graphics (au-dessus de la touche 9). Ceci transforme le curseur en un G, et si vous appuyez sur n'importe laquelle des touches de la première rangée, excepté 9 et 0, vous verrez s'afficher sur l'écran les caractères notés en blanc. En pressant n'importe quelle touche de la première rangée en même temps que Caps Shift, vous verrez les caractères affichés en couleurs inversées, c'est-à-dire le blanc devient noir et vice versa.

Vous pouvez vous étonner de ce qui se passe en appuyant sur n'importe quelle touche de la première rangée, Caps Shift étant maintenue et le curseur étant en mode normal L. Après tout, il n'existe pas de nombres majuscules ! La réponse est qu'en fait aucun nouveau caractère n'est produit. Ceci vous permet d'accéder au reste des fonctions de contrôle montrées en Figure 2.1. Enfoncer les touches 1 et Caps Shift déplace la ligne où est situé le curseur, dans la zone d'entrée de l'écran où vous pouvez l'afficher. Ceci sera développé plus loin dans ce chapitre. Le fait d'appuyer sur les touches 2 et Caps Shift active CAPS LOCK (verrouillage capitales). La première fois que vous l'actionnerez, vous verrez le curseur C là où vous attendiez L, et les lettres que vous taperez apparaîtront sur l'écran en majuscules. Pour revenir aux minuscules, vous devrez presser à nouveau les touches Caps Shift et 2. Vous verrez INV VIDEO (inverse vidéo) au-dessus de la touche 4 ; ceci est également accessible avec la touche Caps Shift, et aura pour effet d'inverser l'affichage de tout nouveau caractère entré à l'écran. Ils apparaissent alors en blanc sur fond noir plutôt qu'en noir sur fond blanc. Pour obtenir à nouveau l'affichage normal, tapez Caps Shift et la touche 3, ce qui donnera TRUE VIDEO (vidéo normale). En utilisant les touches 5, 6, 7 et 8, en même temps que Caps Shift, vous

accéderez à la touche de contrôle du curseur qui vous permettra de le déplacer tout autour de l'écran. Ces touches sont utilisées aussi bien dans les jeux que dans l'édition de textes. Nous avons déjà rencontré les flèches, gauche (5) et droite (8), et nous verrons bientôt les deux autres. Comme il a été dit précédemment, Caps Shift et la touche 9 vous permettent d'entrer en mode graphique signalé par le curseur G. Si vous êtes déjà dans ce mode, cette combinaison de touches vous ramène au mode normal L. Caps Shift et 0 vous donnent la possibilité d'effacer le caractère à gauche du curseur, pendant que vous entrez ou affichez une ligne.

Les touches numérotées ayant une apparence plutôt différente des autres, il est préférable d'en examiner une en détail, avant d'abandonner l'étude du clavier. La figure suivante montre la touche en haut et à gauche (touche 1) et ce qui suit résume la manière d'obtenir tous les mots et symboles qui l'entourent ainsi que les autres touches numériques.

- Lorsque le curseur est K ou L, appuyer sur une touche numérique produira le chiffre correspondant, 1 dans ce cas.

- L'enfoncer en même temps que Symbol Shift produira le symbole en rouge situé en bas et à droite de chaque touche (point d'exclamation dans l'exemple).

- Cette touche plus Caps Shift, a pour effet d'appeler la fonction de contrôle indiquée en blanc au-dessus. La fonction de contrôle associée à chacune des touches numériques, a été détaillée précédemment. Dans le cas de notre touche 1, cela aura pour conséquence d'afficher en bas de l'écran, une ligne du programme en cours en vue de la modifier.

- Lorsque le curseur est E clignotant et que la touche Symbol Shift est maintenue, les commandes imprimées en rouge sur le boîtier du Spectrum sont exécutées. Dans notre exemple, DEF FN permet de définir une fonction.

- Les symboles graphiques surmontant chaque touche, sont obtenus en mode graphique (curseur G). La touche 1 dans ce cas, donne un carré plein dans lequel il manque le cadran droit du haut.

Une autre commodité du clavier mérite d'être mentionnée. Il s'agit de la possibilité de répéter n'importe quel caractère en maintenant la touche enfoncée. Si vous gardez le doigt sur une touche, le curseur étant L clignotant, vous verrez la ligne se remplir de caractères et entendrez un cliquetis venant du Spectrum.

A ce stade, vous devez commencer à penser que le clavier du Spectrum est la chose la plus compliquée que vous ayez jamais rencontrée. En fait, c'est compliqué, mais avec l'expérience, vous trouverez que c'est très logique.

UTILISATION D'UN MAGNETOPHONE

Le prochain programme de démonstration est considérablement plus long que le premier et comme vous auriez des regrets à débrancher la machine après avoir fini de l'entrer et de l'exécuter, et ainsi de le perdre complètement, il est temps maintenant d'apprendre à se servir d'un magnétophone pour sauvegarder et charger les programmes. Le Spectrum peut utiliser presque tous les magnétophones standard, mais il est certain que meilleur il sera, plus le résultat sera fiable. Si vous pensez acheter un magnétophone à cassettes pour votre Spectrum, essayez de prendre un modèle équipé de prises jack miniatures pour l'écouteur et le micro. Si le magnétophone que vous projetez d'utiliser n'en est pas pourvu, vous devrez acheter un

adaptateur dans un magasin de HI-FI, car tous les Spectrum utilisent de petites prises jack miniatures.

Pour connecter le Spectrum au magnétophone, raccordez simplement les deux prises de votre appareil (vous les reconnaîtrez facilement, car ce sont les seules à ne pas être encore utilisées) aux deux prises marquées MIC et EAR au dos du Spectrum. A ce stade, l'ordre des connexions n'a aucune importance. Vous êtes prêt maintenant à enregistrer votre premier programme.

Pour donner quelque chose à sauvegarder au Spectrum, tapez :

```
10 REM ceci est un essai
20 REM ceci est une seconde ligne
30 REM ceci est une troisieme ligne
```

puis tapez SAVE "essai" et pressez la touche ENTER. Le message start tape and press any key (mettez le magnétophone en marche puis appuyez sur n'importe quelle touche) apparaîtra. A ce moment-là, votre Spectrum est prêt à enregistrer, mais il attend que vous lui disiez que le magnétophone est en marche. Montez une bande vierge, réglez le niveau d'enregistrement à la moitié (ceci n'est pas nécessaire si votre magnétophone a un accord automatique de niveau) puis mettez-le en position enregistrement. Lorsque vous presserez une touche, vous verrez apparaître des lignes horizontales en bordure de l'écran ; ceci veut dire que le Spectrum est en train d'enregistrer un programme. Lorsque le message OK apparaît sur l'écran, le programme est sauvegardé sur la bande. Cependant, bien que le programme ait été enregistré, il est encore possible, pour un certain nombre de raisons, qu'il ne soit pas suffisamment bon pour pouvoir être relu par le Spectrum. Par exemple, si vous avez réglé le niveau d'enregistrement trop haut ; ou encore, la bande peut être défectueuse.

Pour s'assurer que le programme a été correctement sauvegardé, rembobinez la bande, débranchez la prise du micro sur le magnétophone et branchez l'autre jack dans la prise d'écoute. Maintenant, tapez :

```
VERIFY "essai"
```

puis pressez ENTER. Mettez en marche le magnétophone ; si le message OK apparaît à l'écran, cela signifie non seulement que le programme sur bande est lisible, mais qu'il est bien identique à celui encore présent dans la mémoire de l'ordinateur. Si quelque chose s'est mal passé, vous verrez apparaître le message R Tape loading error (erreur au chargement de la bande) ou simplement aucun message. Cela signifie que pour une raison quelconque, le Spectrum n'a pas réussi à lire la bande ou n'a même pas pu retrouver l'information. Dans ce cas, la meilleure chose à faire est de rembobiner la bande et de l'écouter (retirez le connecteur de la prise d'écoute du magnétophone, de manière à entendre ce qui s'est passé). Au début de la bande, vous entendrez un son régulier ; il devra être fort mais sans distorsion. Si les sons sont corrects, écoutez une portion non enregistrée de la bande, au même volume. Vous devrez entendre un faible sifflement. Si les sons ressemblent à une pluie d'orage, réduisez le volume et modifiez le contrôle de tonalité puis essayez à nouveau. Si une portion non enregistrée est absolument silencieuse, augmentez le volume.

Après avoir utilisé VERIFY pour contrôler que le programme est correctement sauvegardé, vous pouvez essayer de charger le programme. Débranchez puis rebranchez le Spectrum, pour être sûr que le programme d'essai a bien disparu, puis tapez :

```
LOAD "essai"
```

ou bien

```
LOAD ""
```

et pressez ENTER. La première version de LOAD cherchera sur la bande jusqu'à ce qu'un programme baptisé "essai" soit trouvé, alors que la deuxième version chargera le premier programme de la bande sans tenir compte du nom.

Une fois le magnétophone en marche, ne modifiez le volume et le contrôle de tonalité que si nécessaire.

L'AFFICHAGE ET NOTRE SECOND PROGRAMME

A ce stade, vous devez vous sentir suffisamment confiant pour entrer un programme plus long. Le programme donné ci-après produit le modèle de carte test montré en Figure 2.2. Mis à part le fait de donner un exemple intéressant, vous pourrez l'utiliser pour accorder le téléviseur avec le Spectrum, afin d'obtenir la meilleure définition d'image et les plus belles couleurs. Avant de commencer à taper le programme, les remarques suivantes pourront vous aider. Premièrement, tapez NEW (nouveau) pour effacer tout programme précédent.

Tous les mots, à part ceux tels que "Spectrum TV" en ligne 130 et "Canal 36" en ligne 140, sont obtenus au moyen d'une simple touche, aussi cherchez sur le clavier jusqu'à ce que vous les trouviez ! Remarquez aussi que le signe moins qui précède 175 en ligne 110, est obtenu en pressant la touche J en même temps que la touche Symbol Shift. Du fait de la difficulté d'impression des caractères en mode graphique, une notation spéciale est utilisée. Partout où vous voyez un caractère entre accolades, vous devrez l'entrer en mode graphique. Si le caractère est précédé par une flèche verticale "↑", vous devrez appuyer sur Caps Shift en même temps. Ainsi le "{↑8}" en lignes 170, 180, 190 et 200, signifie "entrez en mode graphique et pressez la touche 8 en même temps que Caps Shift". Le résultat devrait être un bloc noir plein.

```
10 FOR i=0 TO 240 STEP 16
20 PLOT i,0
30 DRAW 0,175
40 NEXT i
50 FOR i=0 TO 175 STEP 16
60 PLOT 0,i
70 DRAW 255,0
80 NEXT i
90 PLOT 0,175
100 DRAW 255,0
```

```
110 DRAW 0,-175
120 CIRCLE 127,81,64
130 PRINT AT 10,6; "S P E C T R U M  T V"
140 PRINT AT 12,11; "Canal 36"
150 FOR c=0 TO 7
160 INK c
170 PRINT AT 3+2*c,2;"[↑8]"
180 PRINT AT 4+2*c,2;"[↑8]"
190 PRINT AT 3+2*c,28;"[↑8]"
200 PRINT AT 4+2*c,28;"[↑8]"
210 NEXT c
220 INK 0
```

Lorsque vous aurez fini d'introduire le programme, tapez simplement RUN, et vous verrez le modèle de la carte test apparaître sur l'écran. Si cela n'apparaît pas tel que sur la Figure 2.2, il est probable que vous ayez commis des erreurs de frappe, vérifiez donc très soigneusement le listing. Appuyez sur la touche LIST afin d'afficher le programme sur l'écran. Vous pouvez également avoir oublié une ligne. Il est très facile de remédier à cela. Pour insérer une ligne, tapez-la avec son numéro de ligne, elle

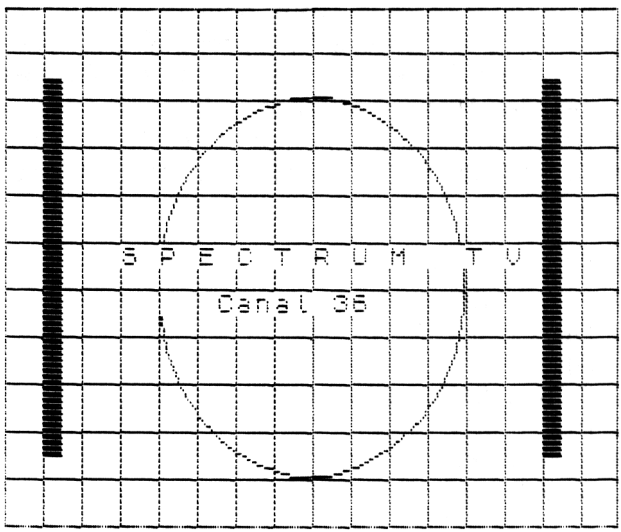


Figure 2.2 : Carte test du Spectrum

apparaîtra en bas de l'écran. Pressez ENTER et elle sera automatiquement insérée à sa place dans le listing. La numérotation des programmes d'ordinateur est normalement faite par pas de dix, ce qui vous permet d'insérer des lignes supplémentaires si besoin est. Si vous souhaitez supprimer une ligne entière, il vous suffit de taper son numéro, puis ENTER pour qu'elle disparaisse, aussi faites très attention de ne pas effacer des lignes que vous désirez conserver. Si vous décelez une erreur sur une ligne, vous n'êtes pas obligé de la retaper entièrement. Au lieu de cela, vous pouvez l'afficher. Le programme étant listé à l'écran, si vous utilisez les touches flèche haute et flèche basse (respectivement 7 plus Caps Shift et 6 plus Caps shift), vous verrez que vous pouvez déplacer le curseur de bas en haut de l'écran, vous positionnant ainsi sur n'importe quelle ligne. Si vous placez le curseur sur une ligne contenant une erreur, puis appuyez sur EDIT (1 plus Caps Shift), vous la verrez réapparaître en bas de l'écran (la zone d'entrée), là où vous l'aviez tapée la première fois. Vous pourrez alors utiliser les touches flèche à droite et flèche à gauche (sur les touches 8 et 5) et la touche DELETE (0 plus Caps Shift) pour corriger vos erreurs. Quand vous aurez fini d'afficher la ligne, appuyez simplement sur ENTER, et la ligne corrigée reprendra sa place dans le programme.

Vous pouvez parfois gagner du temps en entrant vos programmes à l'aide des facilités d'affichage. Par exemple, les lignes 170 à 200 étant très ressemblantes, au lieu d'entrer chaque ligne une par une, entrez la ligne 170 puis (le curseur étant dessus) pressez EDIT. Ceci aura pour effet de copier cette ligne dans la zone d'entrée où vous n'aurez qu'à changer le numéro en 180 et le 3 suivant AT en 4. Si vous pressez alors ENTER, vous verrez que la ligne 180 apparaît dans le programme sans que vous n'ayez eu à la taper.

PREMIERE CONCLUSION

Comme vous apprendrez le BASIC et les principales caractéristiques du Spectrum dans le reste de ce livre, il vous faudra maîtriser correctement le clavier, aussi entraînez-vous jusqu'à saturation ! Toutefois, souvenez-vous de ce qui a été dit au début de ce chapitre, et ne cédez pas à la frustration résultant de fausses manipulations, ce qui pourrait gêner votre compréhension des ordinateurs et du BASIC en particulier.

3

PREMIERS PAS : VARIABLES, PRINT, LET ET INPUT

Un programme est une liste d'instructions que votre ordinateur interprète. Une question se pose : quelles sortes d'instructions peut-on utiliser dans un programme ? Il est clair que des instructions du type : "va me faire une tasse de thé", sont trop générales pour être exécutées par quelqu'un d'autre qu'un être humain ! Les instructions utilisées dans un programme doivent être très précises. Elles doivent spécifier exactement ce qui doit être fait et, peut-être de façon moins apparente, comment cela doit être fait. En d'autres termes, une instruction machine indique à l'ordinateur quoi faire et comment le faire. Dans ce chapitre, nous examinerons les éléments les plus simples de BASIC avec lesquels nous construirons quelques petits programmes.

VARIABLES

Le concept de variable est l'élément le plus important de la programmation. Une variable est une zone mémoire utilisée pour stocker une information. Cela semble très simple et pourtant cela cache une ou deux subtilités. Si vous désirez stocker une information dans une zone mémoire, vous allez avoir besoin d'un moyen pour vous y référer. Il vous faudra lui donner un nom ! Par exemple, dans un cabinet de recrutement, on donne un nom et un seul à chaque fichier. Imaginez la confusion résultant d'une demande relative à un fichier, s'il en existait deux portant le même nom ! Il en va de même en BASIC. Une zone mémoire utilisée pour stocker une donnée, une variable, doit avoir été baptisée d'un nom unique auquel on se réfère. La seule difficulté supplémentaire en BASIC vient du fait qu'il vous faut aussi définir le type de données que vous souhaitez stocker dans la zone mémoire. Une des raisons de cela (nous en verrons d'autres plus tard) est que la réservation de place mémoire dépend du type de données à stocker. Pour le moment, les seules informations que nous manipulerons seront des nombres de toutes sortes. Une variable utilisée pour stocker des nombres est appelée variable numérique ou variable simple.

Vous ne pouvez pas donner à une variable le premier nom qui vous vient à l'esprit, car cela pourrait induire l'ordinateur en erreur. Par exemple, supposez que vous ayez donné le nom "1" à une variable. Comment le Spectrum pourrait-il reconnaître la différence entre la variable 1 et le chiffre 1 ? Dans le cas du Spectrum, vous pouvez donner à une variable simple un nom de n'importe quelle longueur, à ceci près qu'il doit commencer par une lettre et peut se continuer par des lettres et/ou des chiffres. Vous pouvez mélanger les majuscules et les minuscules, mais le Spectrum ne fera aucune différence (le nom de variable "A" est traité de la même manière que "a"). Vous pouvez aussi insérer des espaces n'importe où dans le nom de variable pour le rendre plus clair, mais la machine les ignore. Voici une liste d'exemples corrects de noms de variables simples :

somme, SOMME, Somme
Resultat total, RESULTAT TOTAL, RESULTATTOTAL
Total1
total2
jour2mois3annee80

Remarquez que toutes les versions de "somme" sont traitées de la même manière, ainsi que "résultat total". Il est souvent difficile d'imaginer des noms de variables qui suggèrent la nature de l'information qu'elles doivent contenir, mais il vaut mieux s'y attacher. Si vous relisez un programme longtemps après l'avoir écrit, des noms de variables clairs et évidents vous faciliteront sa compréhension ! Cependant vous devez essayer d'éviter les noms de variables trop longs. Ils sont très pénibles à taper et à retaper dans un programme. Exemples de noms de variables que le Spectrum rejetterait :

nom	motif du rejet
1jour	commence par un chiffre
*date	commence par un * qui n'est pas une lettre
date*	contient un * qui n'est ni une lettre ni un chiffre
reponse?	? n'est ni une lettre ni un chiffre
au-dehors	- n'est ni une lettre ni un chiffre

METTRE DES CHOSES DANS UNE VARIABLE - LET

A présent que nous connaissons des variables et savons comment les baptiser, il est temps de découvrir comment y mettre des informations. Ceci est réalisé en BASIC grâce à l'ordre LET soit, par exemple :

```
10 LET total = 56
```

qui stockera le nombre 56 dans la zone mémoire appelée total. Si vous vous souvenez, les lignes d'un programme sont entrées avec des numéros de ligne qui contrôlent leur séquencement dans la liste d'instructions qui constitue le programme.

Cet exemple est en fait notre premier programme ! Si vous l'entrez tel quel, rien ne se passera jusqu'à ce que vous tapiez RUN quand le Spectrum sera prêt à exécuter la liste d'ordres. Dans l'exemple il n'y a qu'un seul ordre très facile à exécuter. Le nombre 56 est stocké dans une zone mémoire appelée total. En fait il s'est passé d'autres choses. Avant que le programme n'ait été lancé, il n'existait pas de zone appelée total pour y mettre 56. Lorsque le Spectrum rencontre le nom d'une variable dans laquelle vous souhaitez mettre quelque chose, il vérifie si ce nom existe déjà, et si ce n'est pas le cas, il réserve une zone mémoire de taille correcte et mémorise son nom. Ainsi cette simple ligne de programme a deux conséquences : elle crée la variable appelée total et lui attribue la valeur 56.

EXTRAIRE LE CONTENU D'UNE VARIABLE - PRINT

L'exemple du paragraphe précédent est un peu frustrant en ce sens que nous devons admettre que le Spectrum a stocké le nombre 56 dans une variable appelée total. Pour pouvoir le vérifier, ce dont nous avons besoin est un ordre qui indique au Spectrum de trouver la variable et d'en afficher le contenu sur l'écran. L'ordre BASIC qui réalise cela est appelé PRINT. Si vous ajoutez à l'exemple précédent une nouvelle ligne numérotée 20, vous obtiendrez le programme suivant :

```
10 LET total = 56
20 PRINT total
```

Si vous lancez le programme, vous aurez la joie de voir le nombre 56 s'afficher dans le coin en haut et à gauche de l'écran.

A ce stade, il est important que vous compreniez que ce qui s'est produit est le résultat des deux lignes du programme. Plus tard, lorsque vous aurez assimilé le BASIC à une seconde langue, vous comprendrez ce qui se passe sans même y penser, mais pour le moment il est tout aussi facile de lire ces deux lignes de programme et de penser que vous les comprenez parce qu'elles sonnent juste ! Ainsi pour résumer ce que nous avons appris, la première ligne crée une variable appelée total et met 56 dedans. La seconde ligne trouve la zone mémoire correspondant au nom total et affiche son contenu sur l'écran. (Remarquez qu'il serait naturel pour un débutant de penser que PRINT total afficherait le mot total sur l'écran. Ainsi, si vous comprenez déjà que son interprétation est fausse, vous n'êtes plus débutant !)

Pour que l'ordre PRINT puisse s'exécuter, il est nécessaire que le Spectrum trouve la variable à laquelle on se réfère. Si pour une raison ou pour une autre vous essayez d'afficher une variable n'ayant pas été créée, le Spectrum affichera un message d'erreur. Pour vérifier cela, supprimez la ligne 10 de l'exemple précédent (en tapant simplement 10 puis ENTER), puis lancez le programme. Vous devriez être à même de comprendre pourquoi le message variable not found (variable non trouvée) apparaît dans le bas de l'écran. Ceci est votre premier bug !

ARITHMETIQUE

Nos programmes deviennent petit à petit plus intéressants, mais il faut attendre encore un peu avant qu'ils soient opérationnels. Nous pouvons maintenant mettre des nombres dans des variables et en extraire le contenu. Pour que cela soit utile, nous devons être capables de changer le contenu d'une variable et d'afficher quelque chose assimilable à une

réponse. Ceci est assuré par les expressions arithmétiques. Une expression arithmétique n'est rien d'autre qu'une opération arithmétique que vous n'avez pas encore effectuée. Par exemple 3 + 6 est une expression arithmétique qui, résolue, évalue le résultat à 9. Vous pouvez écrire une expression arithmétique à droite du signe = dans une instruction LET, ce qui conduira le Spectrum à évaluer cette expression et à placer le résultat dans la variable. Essayez par exemple :

```
10 LET total = 3 + 6
20 PRINT total
```

Vous verrez le chiffre 9 apparaître dans le coin gauche en haut de l'écran. Comme prévu, le Spectrum a évalué l'expression et mis le résultat dans total.

Comme pour la plupart des opérations à faire avec un ordinateur, il y a des règles à respecter pour écrire correctement les expressions. Vous pouvez utiliser les quatre opérations avec lesquelles vous aviez l'habitude de travailler en arithmétique. L'addition et la soustraction sont identifiées par les symboles habituels + et -, mais la multiplication et la division utilisent respectivement les symboles * et /. La raison pour laquelle on utilise * pour la multiplication au lieu d'une croix est que le symbole traditionnel peut trop facilement être confondu avec la lettre x. Quelques exemples d'expressions arithmétiques correctes :

expression	évalué à
3 + 2	5
3 * 2	6
6 / 2	3
3 + 2 - 4	1
2.1 + 3.3	5.4

Mis à part les quatre opérations usuelles, il y en a deux autres que l'on peut utiliser sur le Spectrum : le signe "-" algébrique, et l'élévation à une puissance. Le signe - algébrique semble pompeux, mais il s'agit en fait du signe - normal mis devant un simple nombre. Par exemple le - dans 3-2 est le signe normal de soustraction, mais le - de -3 est le signe - algébrique. Bien que le même signe soit utilisé dans les deux cas, nous verrons plus tard qu'ils sont traités légèrement différemment. Le signe d'élévation à une puissance est ↑ . Par exemple 2↑2 est lu comme 2 élevé à la puissance 2, ou 2 au carré soit 4. Le signe d'élévation à puissance n'est pas souvent utilisé, et il est mentionné ici pour être exhaustif plus que pour son importance.

COMPRENDRE LES EXPRESSIONS - L'ORDRE D'EVALUATION

Bien que le concept d'expression arithmétique semble clair, il y a une complication cachée. Par exemple, si vous écrivez l'innocente expression 3+2*4, cela veut-il dire 3+2 (5) fois 4 (réponse 20) ou bien 3 plus le résultat de 2 fois 4 (soit 3 + 8 = 11) ? Cela peut vous paraître étrange qu'il y ait deux manières possibles de traiter cette expression car vous devez deviner qu'une des deux méthodes est évidemment correcte alors que l'autre ne l'est pas. Cependant, même en arithmétique, il n'y a pas de réponse dans l'absolu ! L'interprétation correcte est purement conventionnelle. La question de savoir si nous devons exécuter "+" ou "*" en premier dans une expression, est régie par la régle générale qui donne priorité à la multiplication par rapport à l'addition. De ce fait, la multiplication doit être traitée en premier, ce qui donne le résultat 11 dans notre exemple. L'assignation de priorité est étendue à toutes les opérations utilisées dans une expression (y compris celles que nous n'avons pas encore rencontrées). Les priorités que le Spectrum utilise pour sélectionner l'ordre dans lequel les opérations seront traitées, sont :

opérateur	priorité
↑	10 la plus haute
- algébrique	9
*,/	8
+,-	6 la plus basse

La raison pour laquelle les priorités démarrent à 10 et se terminent à 6, est que d'autres opérateurs qu'il nous reste à voir ont aussi des priorités assignées. Pour évaluer une expression, vous devez toujours commencer par les opérateurs de plus haute priorité. Si à l'intérieur d'une expression, deux opérateurs ont la même priorité, vous traiterez en premier celui de gauche. (En l'absence de tout autre critère, vous traitez de gauche à droite.)

Tout ceci peut vous sembler bien compliqué pour traiter une simple petite opération, mais cela est nécessaire si vous voulez écrire des expressions sans aucune ambiguïté. Cependant il existe un autre moyen de spécifier l'ordre d'évaluation qui permet de passer outre les priorités : les parenthèses ( ). C'est une convention déjà ancienne qui indique que les parties d'expression entre parenthèses, doivent être traitées en premier. Par exemple 3+2*4 donne 11, et (3+2)*4 donne 20. Si jamais vous avez un doute sur l'ordre dans lequel le Spectrum évalue une expression, mettez des parenthèses autour des parties que vous souhaitez voir traitées en premier. Les parenthèses font parfois perdre du temps et demandent des efforts, mais elles lèvent toute ambiguïté.

VARIABLES ET CONSTANTES - L'EXPRESSION COMPLETE

Jusqu'ici nous avons examiné des cas ne faisant appel qu'à des nombres, mais il n'y a aucune raison pour que nous ne puissions pas utiliser des variables dans les expressions. Si vous écrivez une expression telle que TOTAL+3, le Spectrum ira chercher la variable appelée total et trouvera

le nombre qu'elle contient, après quoi il ajoutera 3 à ce nombre. Par exemple, si total contenait 32, l'expression TOTAL+3 serait évaluée à 35. Vous remarquerez que rien ne laisse supposer que la valeur contenue dans TOTAL ait été modifiée. Cette valeur a simplement été utilisée dans l'évaluation de l'expression. Un nombre tel que 32 est appelé constante (parce que sa valeur ne change jamais), et maintenant nous voyons qu'une expression peut être composée de variables et de constantes, combinées avec les opérateurs arithmétiques +, -, /, *, ↑. Une expression résulte toujours en une constante, et c'est celle-ci qui est stockée dans une variable par l'instruction LET.

UN PETIT PROGRAMME

En utilisant tout ce que nous avons rencontré jusqu'à présent sur les constantes, variables et expressions, nous pouvons maintenant écrire un petit programme qui additionne deux nombres :

```
10 LET nombre1=23.34
20 LET nombre2=44.32
30 LET reponse=nombre1+nombre2
40 PRINT reponse
```

Si vous entrez puis lancez ce programme, vous verrez que la somme des deux nombres des lignes 10 et 20, est affichée par la ligne 40. Bien que ce soit un exemple simple, cela montre une grande variété d'idées de programmation. En lignes 10 et 20 l'instruction désormais familière LET est utilisée pour stocker deux constantes dans deux variables. En ligne 30 l'expression 'nombre1+nombre2' est évaluée et le résultat est placé dans une troisième variable 'réponse'. La ligne 40 affiche le contenu de 'réponse' sur l'écran. Si vous trouvez cela facile, c'est parfait ! Essayez de changer les lignes 10 et 20 pour additionner différents nombres entre eux, et modifiez la ligne 30 pour vous donner d'autres expressions arithmétiques.

UN AUTRE MOYEN DE MODIFIER LES VARIABLES - INPUT

Dans le précédent exemple, les deux variables 'nombre1' et 'nombre2' contenaient des valeurs ayant été entrées à l'aide de l'instruction LET. Ceci est pratique, sauf si nous voulons utiliser le programme plusieurs fois avec différentes valeurs. Comme il a déjà été dit, le seul moyen de modifier le contenu des variables est de modifier chaque ligne du programme avant de le lancer. Evidemment, ce dont nous avons besoin est une instruction qui nous permettrait d'entrer n'importe quelle valeur dans les variables pendant l'exécution du programme. C'est ce que fait l'instruction BASIC 'INPUT'. Essayez par exemple le programme suivant :

```
10 LET nombre1=5
20 INPUT nombre2
30 LET reponse=nombre1+nombre2
40 PRINT reponse
```

Lorsque vous allez lancer ce programme, vous allez être surpris de constater que rien ne se passe ! Restez calme ! Ce qui s'est produit est que la ligne 10 a été exécutée, ce qui a eu pour effet de mettre 5 dans la variable nombre1. Puis le Spectrum se déplace à la ligne 20, où il obéit à l'ordre INPUT, en attendant de recevoir un nombre. C'est pourquoi rien ne se passe. L'ordinateur attend en effet que vous lui donniez un nombre, puis que vous appuyez sur ENTER pour signaler que vous avez fini d'entrer ce nombre. Il stocke alors le nombre que vous lui avez donné, dans la variable 'nombre2', puis passe à l'instruction suivante. De la sorte, si vous ne l'aviez pas déjà fait, lancez le programme et tapez le nombre de votre choix. Vous avez le plaisir de voir votre nombre, additionné à 5, affiché à la place habituelle sur l'écran.

Nous disposons maintenant de deux manières de mémoriser des nombres dans des variables : LET et INPUT. Il est important de comprendre la différence entre ces deux ordres. Comme dans le cas de LET, si une variable n'existe pas lors

de l'exécution d'un ordre INPUT, le Spectrum la créera. En résumé, si vous voulez toujours utiliser la même valeur, ou résultat d'une expression, utilisez LET. Si par contre, vous souhaitez stocker une valeur différente dans une variable à chaque fois que vous lancez le programme, utilisez l'ordre INPUT.

VARIABLES ET CONSTANTES COMME EXPRESSIONS

Une des caractéristiques les plus importantes de BASIC est que presque partout où vous pouvez utiliser une constante ou une variable, vous pouvez aussi bien utiliser une expression. Par exemple, dans l'instruction PRINT vous pouvez écrire :

```
30 PRINT nombre1+nombre2
```

dans laquelle le Spectrum évaluera l'expression et affichera le résultat.

Ceci est d'autant plus vrai que les formes les plus simples d'une expression peuvent être des constantes ou des variables. Par exemple, le nombre3 peut être assimilé à une constante ou une expression extrêmement simple. De même, la variable 'total' peut être identifiée comme une expression. Ainsi non seulement vous pouvez utiliser une expression là où vous devriez prendre une variable ou une constante, mais la réciproque est vraie aussi ! Par exemple :

```
LET nombre1=nombre2
```

et

```
PRINT 3
```

sont deux instructions BASIC correctes.

DESCRIPTION DE BASIC

Il est difficile de décrire une langue, et le BASIC en est une. Le problème vient du fait qu'il est facile de donner un exemple de ce qui est correct, mais en revanche il est difficile d'expliquer toutes les variantes possibles. Par exemple, au début de ce chapitre, l'instruction LET a été introduite par :

```
LET total=56
```

mais ceci n'a pas fait allusion au fait que nous aurions pu écrire la chose sous la forme :

```
LET total=nombre
```

ou

```
LET total=nombre1+nombre2
```

Pour essayer de contourner cette difficulté, on a pour habitude de donner une définition indiquant les formes générales que peuvent prendre les choses. Par exemple la forme générale de l'instruction LET peut être écrite de la façon suivante :

```
LET <variable simple>=<expression arithmetique>
```

dans laquelle les éléments entre crochets ne sont pas pris tels quels mais remplacés par l'un des exemples traités. Ainsi dans une instruction réelle, LET <variable simple> serait remplacée par un nom de variable tel que total, somme, nombre, etc.

Dans tout le reste de ce livre, les instructions BASIC sont introduites par des exemples puis définies de la même façon que l'instruction LET précédente. Au fur et à mesure que nous en apprendrons, il sera nécessaire de redéfinir les

instructions en développant leurs caractéristiques plus en détail. Dans cette optique, les deux autres instructions BASIC que nous avons introduites, PRINT et INPUT, peuvent être définies ainsi :

PRINT <expression arithmetique>

et

INPUT <variable simple>

mais comme nous le verrons plus tard, ce ne sont pas les définitions finales !

INPUT AVEC MESSAGE

Bien que nous puissions maintenant utiliser INPUT pour mémoriser des informations dans des variables, le fait que le Spectrum attende que quelqu'un tape un nombre pour continuer, est un peu gênant. Ce qu'il faut, c'est la possibilité d'afficher sur l'écran un message du type "tapez un nombre maintenant" ou bien "quel est votre nombre". Le BASIC offre deux possibilités pour afficher de tels messages. Essayez le petit programme suivant :

```
10 PRINT "ceci est un message"
20 INPUT "quel est votre nombre? ";nombre1
```

La ligne 10 affichera "ceci est un message" en haut et à gauche de l'écran, et la ligne 20 "quel est votre nombre?" en bas de l'écran et attendra que vous entriez un nombre. Dans les deux cas, les caractères affichés sont ceux figurant entre guillemets. Les caractères mis entre guillemets forment ce que l'on appelle une chaîne de caractères ou plus simplement une chaîne. Vous pouvez aussi bien utiliser PRINT que INPUT pour afficher des messages sur l'écran, le choix dépendant seulement de ce qui vous semble

le plus pratique. Comme exemple d'utilisation simultanée, regardez la version suivante du programme d'addition donné précédemment :

```
10 PRINT "quel est votre premier nombre?"
20 INPUT nombre1
30 INPUT "quel est votre deuxieme nombre? ";
   nombre2
40 PRINT nombre1+nombre2
```

PRINT COMBINES

Le fait de pouvoir afficher des messages sur l'écran est une commodité très largement utilisée dans d'autres cas que celui d'une entrée de données. Par exemple, dans le dernier programme il aurait été mieux de préciser que le nombre affiché résultait de l'addition du nombre1 avec le nombre2. Vous pouvez en effet utiliser une simple instruction PRINT pour afficher plusieurs choses à la fois. Par exemple, remplacez la ligne 40 du programme précédent par :

```
40 PRINT nombre1;"+";nombre2;"=";
   nombre1+nombre2
```

et vous verrez apparaître le contenu du nombre1 suivi d'un espace, du signe +, d'un autre espace, puis le contenu du nombre2, le signe = précédé et suivi d'un espace, et enfin le résultat. Vous pouvez considérer cette séquence PRINT comme une liste d'éléments à afficher, chacun étant séparé par un point-virgule et affiché côte à côte. Notre définition générale de PRINT peut être mise à jour sous la forme :

PRINT <liste d'affichage>

dans laquelle <liste d'affichage> est une liste d'éléments séparés par des points-virgules. Chaque ordre PRINT demande

l'affichage sur une nouvelle ligne d'écran. Dans le Chapitre 7 nous reviendrons sur la définition de la <liste d'affichage> en y incluant les moyens de former les informations sur l'écran, mais d'ici là cette version de <liste d'affichage> satisfera tous vos besoins.

QUELQUES PROGRAMMES-EXEMPLES

Avec si peu de BASIC, il est tout de même possible d'écrire des programmes utilisables, à condition qu'ils soient simples. Par exemple, si vous aviez besoin de connaître régulièrement combien vous pourriez acheter de dollars avec une somme en francs, un programme de conversion serait utile. La définition de ce programme peut se faire facilement comme suit :

demander à l'utilisateur le taux de conversion
demander le nombre de francs à convertir en dollars
afficher le résultat

Vous comprenez que ce programme est vraiment très simple pour votre Spectrum, aussi, avant de regarder le programme donné plus loin, essayez d'en écrire votre propre version. Souvenez-vous des INPUT avec message et des messages explicatifs. Il n'existe aucune manière parfaite d'écrire un programme, aussi ne vous tourmentez pas si votre version diffère de celle qui est proposée. Il se pourrait qu'elle fonctionne mieux !

```
10 PRINT "conversion de Francs en Dollars"
20 INPUT "quel est le taux de conversion? ";
   taux
30 INPUT "combien de Francs voulez-vous
   depenser? ";montant
40 PRINT
```

```
50 PRINT "pour ";montant;" Francs"
60 PRINT "vous pouvez acheter ";taux*montant;
   " Dollars"
```

La ligne 10 affiche simplement le titre du programme. Les lignes 20 et 30 suggèrent et acceptent les entrées nécessaires. Remarquez que la ligne 40 est utilisée pour sauter une ligne sur l'écran, afin de séparer l'entrée des données du résultat. Les lignes 50 et 60 affichent la réponse avec quelques explications.

Autre exemple, considérez le problème de calcul de la distance d'arrêt d'un véhicule en fonction de sa vitesse en Km/h. Le point le plus important pour écrire ce programme est de connaître la façon de calculer la distance d'arrêt. Il est important de réaliser, à ce stade, qu'aucun ordinateur ne peut calculer quelque chose sans que vous lui ayez expliqué comment effectuer les calculs. Le code de la route révèle que la distance d'arrêt est constituée de deux composantes - une distance de ralentissement et une longueur de freinage. La valeur de la distance de ralentissement en mètres est à peu près la même que la vitesse en Km/h. La longueur de freinage est un peu plus compliquée et est donnée par le carré de la vitesse divisé par 20. Cependant ces éléments sont vraiment très faciles à calculer pour le Spectrum ; aussi pour rendre plus pratique ce petit programme, il est raisonnable d'afficher la distance de ralentissement, la distance de freinage, la marge de sécurité et le nombre de longueurs de voiture. On prendra comme longueur moyenne de voiture 4 m. Le programme est maintenant facile à écrire :

```
10 PRINT "distance d'arret"
20 PRINT
30 INPUT "vitesse en Km/h = ";vitesse
40 PRINT "a ";vitesse;" Km/h"
50 PRINT
60 PRINT "distance de ralentissement = ";vitesse;"m"
```

```
 70 LET distfrein=vitesse*vitesse/20
 80 PRINT "distance de freinage = "distfrein;" m"
 90 PRINT "marge de securite = ";
    vitesse+distfrein;" m"
100 PRINT
110 PRINT "qui represente ";(vitesse+distfrein)/4;
    " longueurs de voiture"
```

La ligne 10 affiche seulement le titre du programme. La ligne 30 suggère l'entrée de la seule information nécessaire au programme : la vitesse en Km/h. Elle est stockée dans la variable 'vitesse'. Les lignes 40 et 50 commencent à donner la réponse à l'utilisateur en affichant la vitesse pour laquelle le calcul s'opère, et en laissant une ligne blanche. La ligne 60 affiche la distance de ralentissement en m, qui ne requiert aucun calcul, puisque celle-ci est numériquement identique à la vitesse. La distance de freinage est calculée en ligne 70 et affichée par la ligne 80. La distance totale est calculée et affichée en 90. Remarquez que cette ligne représente un exemple d'expression arithmétique dans une instruction PRINT. La ligne 110 calcule et affiche la distance totale exprimée en nombre de longueurs de voiture. Notez que vous devez mettre l'addition entre parenthèses pour être certain qu'elle sera traitée en premier.

Le dernier exemple de ce chapitre est un programme concret, qui peut réellement être utilisé. Un problème très courant est l'estimation des quantités de sable, gravier et ciment que vous devriez acheter pour construire une dalle en béton. Le programme suivant qui est "un calcul de béton" peut aider le résoudre. Encore une fois, notre première préoccupation consiste à savoir comment calculer avant de commencer à écrire le programme. L'examen d'un manuel de construction donne les éléments suivants : pour un mètre-cube de béton constitué d'une part de ciment, **x** parts de sable, et **y** parts de gravier, il faut :

$$\frac{1}{0.025*(1+x+y)}$$

sacs de ciment et

$$\frac{x*1.5}{(1+x+y)}$$

mètre cube de sable et

$$\frac{y*1.5}{(1+x+y)}$$

mètre cube de gravier. (Il n'est pas important de comprendre comment ces équations sont établies, il est en effet très fréquent qu'un bon programme soit écrit à partir d'informations que le programmeur ne comprend pas complètement, et pourquoi en serait-il autrement !)
Transformer ces équations sous forme de programme est, là encore, une chose relativement facile. La première partie du programme devra demander les dimensions de la dalle de béton puis en calculer son volume. Ensuite le programme demandera le ratio de mélange, puis en utilisant les équations données plus haut, calculera le nombre de sacs de ciment et le volume de sable et de gravier nécessaire. Enfin les résultats devront être affichés sous une forme satisfaisante pour l'utilisateur. Les détails du programme sont :

```
10 PRINT "CALCUL DE BETON"
20 PRINT
30 INPUT "quelle est l'epaisseur en mm? ";epais
40 INPUT "quelle est la longueur en m? ";long
50 INPUT "quelle est la largeur en m? ";largeur
60 PRINT
70 LET vol=epais*.001*long*largeur
80 PRINT "volume total = ";vol;" en m cube"
```

```
90 PRINT
100 INPUT "combien de parts de sable pour une
    part de ciment? ";parsable
110 INPUT "combien de parts de gravier pour une
    de ciment? ";pargrav
120 LET total=1+parsable+pargrav
130 LET ciment=vol/total/0.025
140 LET sable=vol*parsable*1.5/total
150 LET gravier=vol*pargrav*1.5/total
160 PRINT "Pour le melange de rapport 1:";
    parsable;":";pargrav;"melange"
170 PRINT "Vous avez besoin de"
180 PRINT ciment;" sacs de ciment"
190 PRINT sable;" m cube de sable"
200 PRINT "et ";gravier;" m cube de gravier"
```

Les lignes 30 à 80 demandent les dimensions de la dalle, calculent et affichent simultanément le volume total. Les lignes 100 et 110 demandent le ratio pour le mélange, et la ligne 120 évalue 1+**x**+**y** utilisé dans toute la suite des calculs. Les lignes les plus intéressantes du programme sont celles de 130 à 200. Les lignes de 130 à 150 effectuent les calculs principaux, et la section suivante (160 à 200) affiche les résultats. Si vous regardez attentivement les calculs, vous observez qu'une expression arithmétique en BASIC ne s'exprime pas toujours comme l'équation dont elle est issue. Par exemple, vous pourriez tomber dans le piège qui consiste à écrire :

$$\frac{1}{0.025*(1+\mathbf{x}+\mathbf{y})} \quad \text{comme} \quad 1/0.025*(1+\mathbf{x}+\mathbf{y})$$

dont le résultat de cette expression arithmétique BASIC est obtenu en divisant 1 par 0.025 puis en multipliant le résultat par (1+**x**+**y**) ; ce qui reviendrait à :

$$\frac{1}{0.025} *(1+\mathbf{x}+\mathbf{y})$$

plutôt que l'équation à laquelle nous nous intéressions. L'expression correcte est :

```
1/0.025/(1+x+y)
```

ou si vous préférez utiliser davantage de parenthèses pour lever toute ambiguïté :

```
1/(0.025*(1+x+y))
```

Dans cette section qui affiche le résultat, remarquez le moyen par lequel chaque réponse est insérée dans le message affiché. Souvenez-vous qu'il n'existe aucune règle standard pour produire les résultats, seule l'expérience vous guidera.

4

BOUCLES ET CHOIX FLUX DE CONTROLE

Précédemment, nous avons écrit des programmes représentés par une liste d'instructions exécutées une par une et de haut en bas. Bien qu'il soit possible d'écrire des programmes opérationnels sans rien utiliser de plus, la programmation ne devient intéressante que lorsque vous pouvez modifier l'ordre dans lequel sont exécutées les instructions.

LE FLUX DE CONTROLE OU SEQUENCE D'EXECUTION

Si vous regardiez les exemples de programmes à la fin du Chapitre 3, il vous serait possible de suivre avec le doigt l'ordre dans lequel le Spectrum exécuterait les instructions. Cela peut vous faire penser au tracé d'un flux de contrôle à travers le programme. Chaque instruction à son

tour gère ou vérifie ce que fait le Spectrum et passe le contrôle à l'instruction suivante. En l'absence de toute autre information, la prochaine instruction est prise dans la ligne suivante. Donc le flux de contrôle par défaut est une ligne partant du début du programme et finissant en bas. Ce qui suit est un programme très simple illustrant cette condition par défaut.

```
10 INPUT a
20 PRINT a
30 INPUT b
40 PRINT b
```

Figure 4.1

BOUCLES - L'INSTRUCTION GOTO

Le BASIC prévoit une simple instruction pour modifier le flux de contrôle : l'ordre GOTO (aller à). Essayez le programme suivant :

```
10 LET test=0
20 PRINT test
30 GOTO 20
```

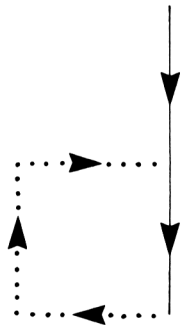


Figure 4.2

Si vous faites fonctionner ce programme, vous verrez l'écran se remplir de zéros (un à chaque ligne) puis le

Spectrum s'arrêtera et demandera "Scroll ? y/n" (défilement ? oui/non). Si vous répondez oui à la question, l'écran se couvrira à nouveau de zéros et ainsi de suite. Ce programme est notre premier exemple de boucle. En traçant le flux de contrôle à travers le programme, vous voyez pourquoi le mot boucle est approprié. La ligne 10 mémorise zéro dans la variable 'test', puis la ligne 20 imprime le contenu de 'test'. La ligne 30 est nouvelle pour nous en ce sens qu'elle utilise GOTO, mais sa signification est claire simplement en la lisant. L'ordre GOTO 20 indique au Spectrum d'exécuter la ligne 20 comme instruction suivante. Ainsi, après la ligne 30 le contrôle est transféré à la ligne 20 et le contenu de la variable 'test' est imprimé sur l'écran une seconde fois ; puis cette ligne 30 est à nouveau exécutée. Ceci transfère le contrôle à la ligne 20 et ainsi de suite... jusqu'à ce que l'écran soit rempli de zéros et que le Spectrum demande la permission d'effacer l'écran pour faire de la place afin d'afficher les mêmes zéros ! La répétition des lignes 20 et 30 continuera indéfiniment et le tracé du flux de contrôle montre qu'il prend la forme d'une boucle. Cependant ce type de boucle n'est pas très utilisé parce que la seule façon de l'arrêter est d'appuyer sur la touche Break (arrêt) ou répondre non à la question "Scroll ? y/n" ; on l'appelle boucle infinie.

Un ordre GOTO peut être utilisé pour obliger le Spectrum à exécuter n'importe quelle instruction. Sa forme générale est :

GOTO <expression arithmetique>

vous pouvez donc écrire par exemple :

GOTO 2*10 (aller a 2*10)

ayant la même signification que GOTO 20 (aller à 20) ; cependant ce type d'expression n'est pas utilisé très souvent, et il est préférable de penser à GOTO comme :

GOTO <numéro de ligne>

Vous pourriez être surpris de ce qui arrive en écrivant GOTO 50 (aller à 50) si la ligne 50 n'existe pas. Plusieurs versions de BASIC vous arrêteraient simplement en affichant un message d'erreur du fait que vous avez essayé de transférer le contrôle à une ligne inexistante, mais la version Spectrum du BASIC est un peu différente. Si le numéro de ligne n'existe pas, alors le contrôle est transféré à la ligne dont le numéro est immédiatement supérieur. Par exemple si vous utilisez GOTO 50 et que la ligne n'existe pas, mais que la ligne 55 existe, alors le contrôle passera à la ligne 55. S'il n'y a pas de "prochaine ligne supérieure", GOTO essaie de transférer le contrôle hors du programme, et le programme s'arrête sans message d'erreur. Cela signifie qu'il est virtuellement impossible de faire une erreur en utilisant GOTO ! Vous pouvez penser que c'est un avantage, mais faites très attention parce que si vous avez fait une erreur de frappe dans un GOTO, le Spectrum ne vous dira pas que vous transférez le contrôle à une ligne inexistante. Ceci peut ne pas poser de problème au début parce que le numéro de ligne immédiatement supérieur peut justement être la ligne où vous vouliez de toute façon aller, mais si vous insérez des lignes dans le programme par la suite, le problème pourra apparaître à ce moment-là !

Bien que l'exemple de boucle infinie serve à introduire la notion de transfert du contrôle à un point différent du programme, il n'indique pas réellement le type de traitement pour lequel on utilise une boucle. Un concept important en programmation est la répétition d'opérations en série. Ceci est souvent appelé itération. Par exemple, l'instruction LET total=total+1 ajoute simplement 1 au contenu de la variable 'total' et remémorise le résultat dans 'total'. En d'autres termes, il incrémente de 1 le nombre stocké dans 'total'. Si vous répétez cette opération en utilisant GOTO, vous obtenez plus qu'un simple ajout de 1 à la variable ; vous obtenez un programme de calcul ! Essayez le programme suivant :

```
10 LET total=0
20 LET total=total+1
30 PRINT total
40 GOTO 20
```

Vous verrez l'écran rempli de nombres de 1 à 21 et l'habituelle question "Scroll ? y/n". En répondant oui vous pouvez faire défiler les nombres pendant très longtemps.

Notez que par addition de GOTO à une instruction qui ajoute 1 à une variable, nous avons l'impression d'obtenir un programme qui fait un peu plus que compter. En fait, nous générons une séquence de nombres. Et ceci commence à ressembler davantage à de la programmation réelle, que les programmes du type "une instruction après l'autre" du Chapitre 3. Pour voir que les boucles peuvent faire des choses un peu plus utiles, essayez :

```
10 LET total=0
20 PRINT "x=";total;" x au carre=";total*total
30 LET total=total+1
40 GOTO 20
```

Ce programme affichera deux types de nombres, le second étant le carré du premier.

Même si vous êtes familiarisé avec le concept de boucle, vous pouvez vous interroger sur la valeur courante d'une variable, en n'importe quel point de la boucle. Par exemple, dans le cas du programme précédent, la première valeur affichée de 'total' était 1 alors que dans celui-ci la première valeur est 0. Cette différence est simplement due au fait qu'elle dépend de l'endroit du programme où est placée la ligne qui additionne un 1 à total, et aussi de la valeur à laquelle 'total' est initialisé avant le début de la boucle. Si vous changez la ligne 10 dans le dernier programme, en 10 LET total=1, la première valeur imprimée sera un 1.

Comprendre comment travaille une boucle devient plus facile avec l'expérience. Il faut simplement suivre avec précision et sans précipitation le déroulement du programme.

CHOIX ET CONDITIONS - L'INSTRUCTION IF

Bien que GOTO soit une relation très employée, la seule utilisation que vous puissiez en faire est de former une boucle infinie. Ceci est bien pour des réalisations simples comme imprimer des tables de valeurs, mais c'est un peu trop sommaire pour bon nombre d'applications. Ce qui nous manque est une instruction qui arrête la boucle dès qu'une condition est satisfaite. Par exemple, supposez que nous voulions écrire un programme qui additionnerait dix nombres puis afficherait le résultat. Actuellement, le mieux que nous puissions faire est de taper chaque nombre, puis les additionner dans un total partiel que nous imprimerions à chaque fois, comme dans l'exemple suivant :

```
10 LET total=0
20 LET compte=0
30 INPUT "nombre = ";nombre
40 LET total=total+nombre
50 LET compte=compte+1
60 PRINT compte;" total=";total
70 GOTO 30
```

Pour arrêter ce programme, tapez STOP quand il demandera le nombre suivant. Si vous exécutez ce programme vous constaterez qu'il affiche beaucoup plus de données (ou d'éléments) que nécessaire. Ce que nous voulons faire en fait, c'est lire les dix nombres, conserver la somme partielle et imprimer seulement le résultat final. Ceci peut être réalisé en utilisant la relation IF (si) :

```
10 LET total=0
20 LET compte=0
30 INPUT "nombre = ";nombre
40 LET total=total+nombre
50 LET compte=compte+1
60 IF compte=10 THEN GOTO 80
70 GOTO 30
80 PRINT compte;" total=";total
```

La seule différence entre ce programme et le précédent est l'utilisation de la relation IF en ligne 60. L'ordre IF est exécuté à chaque passage dans la boucle formée par les lignes 30 à 70. Ceci prend la forme d'une comparaison entre le contenu de la variable 'compte' et 10. Si 'compte' n'est pas égal à 10, le contrôle passe à l'instruction suivante, c'est-à-dire à la ligne 70 et l'ordre GOTO suivant THEN (alors) reste sans effet. Cependant, si 'compte' est égal à 10, alors GOTO est exécuté et le contrôle passe à la ligne 80 qui affiche le résultat et termine la boucle.

Il y a plusieurs manières d'utiliser la relation IF...THEN GOTO pour modifier le flux de contrôle en fonction d'une condition vraie ou fausse. Avant de chercher les différentes sortes de choses qui peuvent être faites avec IF, nous devons examiner les types de conditions utilisables.

Toutes les conditions que nous pouvons utiliser dans une relation IF prennent une forme très simple :

<expression arithmetique1><relation><expression arithmetique2>

Nous savons déjà ce qu'est une expression arithmétique. L'élément nouveau est la relation. En BASIC, il y a six relations :

relation	signification
=	égal
>	supérieur à
<	inférieur à
<=	inférieur ou égal à
>=	supérieur ou égal à
<>	différent de

Notez que sur le Spectrum chacun de ces symboles est obtenu par une simple touche. Vous feriez une erreur en tapant un < et un > pour obtenir un simple <> ! La signification en BASIC de chacune de ces relations est la même que leur signification normale.

Ces conditions sont très souvent appelées expressions conditionnelles et une condition est comparable à une expression dans laquelle on évalue une valeur, mais dans ce cas il y a seulement deux résultats possibles, vrai ou faux. Il est important de lire les conditions d'une manière exacte pour éviter les confusions. Par exemple, la condition 'compte=3' n'est pas une instruction qui rend 'compte' égal à 3, c'est une question sur le contenu de 'compte'. Si la valeur mémorisée dans 'compte' est 3, alors 'compte=3' est vrai, mais si la valeur est autre que 3, alors 'compte=3' est faux. Ainsi le Spectrum utilise le signe = de deux manières différentes, comme instruction pour mémoriser une valeur dans une variable, et comme relation dans une condition. Le seul moyen de dire quelle est la signification exacte est dans tous les cas de regarder le reste de l'instruction. Quelques exemples de conditions :

compte <>4	faux, si le compte est 4, vrai sinon
compte*2>10	vrai si compte*2 est plus grand que 10, faux autrement
3>1	TOUJOURS vrai
1>6	TOUJOURS faux

Pour renforcer l'idée qu'une condition est une expression qui peut prendre l'une des deux valeurs (vrai ou faux), il vaut mieux s'imaginer que le Spectrum représente de la même manière les valeurs et les nombres. Vrai est assimilé à 1 et faux à 0. Pour prouver que c'est bien le cas, essayez le programme suivant :

```
10 INPUT "premier nombre";a
20 INPUT "second nombre";b
30 PRINT "le resultat de";a;">";b;
   "est ";a>b
40 GOTO 10
```

Vous verrez qu'un 1 ou un 0 s'afficheront suivant que a>b, dépendant des nombres que vous aurez entrés, sera vrai ou

faux. Vous devriez être surpris de pouvoir écrire une condition dans la relation PRINT alors que vous auriez dû écrire normalement une expression arithmétique. Ceci est simplement dû au fait qu'une condition est semblable à une expression arithmétique et, à ce titre, elle peut être utilisée partout où vous auriez écrit une expression arithmétique. En effet, vous pouvez mélanger les expressions conditionnelles et arithmétiques sans problème tant que vous prêterez attention à l'ordre dans lequel elles seront évaluées. Les relations ont une priorité 4, ce qui signifie que tous les calculs sont effectués avant qu'elles n'aient été évaluées. Par exemple :

```
PRINT (1>2)+(3>1)+(3=2+1)
```

affichera 2 sur l'écran parce que (1>2) est faux et donc estimé à 0, (3>1) est vrai et estimé à 1 et (3=2+1) est vrai aussi, donc estimé à 1, ce qui donne 0+1+1 c'est-à-dire 2. (Notez que toutes ces conditions ont été incluses entre parenthèses pour empêcher le Spectrum d'effectuer les calculs arithmétiques en premier !) Ce processus relève déjà d'une connaissance avancée du BASIC, aussi ne vous inquiétez pas si vous ne comprenez pas tout. Ce qui est important est de bien comprendre que quelque chose comme 2=3 est une expression qui prend la valeur vrai ou faux, et non une instruction qui effectue une opération.

Nous avons fait une légère digression sur l'ordre IF pour examiner le concept d'expression conditionnelle, mais armé de cette nouvelle information le reste de ce chapitre vous semblera plus facile. La forme générale de la relation IF...THEN GOTO est :

```
IF <expression conditionnelle> THEN GOTO
   <numero de ligne>
```

Si l'expression conditionnelle vaut 1 (vrai), alors GOTO est exécuté. Si l'expression conditionnelle vaut 0 (faux), l'instruction suivante est exécutée.

Il y a donc plusieurs façons d'utiliser IF...THEN GOTO. Mis à part les sorties de boucles, il est difficile de donner d'autres exemples. Mais si vous avez compris la notion de flux de contrôle et la façon d'utiliser IF...THEN GOTO pour le modifier, alors vous ne devriez pas avoir de problèmes pour comprendre les exemples de ce livre.

UTILISATION DE IF

Le seul moyen de découvrir comment se servir de IF est d'écrire vos propres programmes en l'utilisant. De la sorte, vous saisirez progressivement toutes les manières standard de modifier le flux de contrôle en fonction du résultat d'une expression conditionnelle. Cependant pour accélérer ce processus et éviter les utilisations maladroites de la relation IF il peut être utile de donner quelques exemples des cas les plus fréquents.

Une relation IF peut être utilisée pour sauter une section de programme.

```
10 INPUT a
20 IF a>0 THEN GOTO 40
30 LET a=-a
40 PRINT a
```

Ce petit programme refuse d'afficher un nombre négatif ! La relation IF en ligne 20 vérifie si le nombre dans 'a' est supérieur à zéro et s'il en est ainsi, le contrôle passe à la ligne 40 en sautant la ligne 30. Si a>0 est faux, la ligne 30 change le signe du contenu de 'a'. Dans un autre programme, une liste différente d'instructions pourrait être sautée, suivant d'autres conditions. La forme du saut est montrée en Figure 4.3 qui illustre comment le résultat d'une condition décide si oui ou non la liste d'instructions doit être exécutée ou sautée.

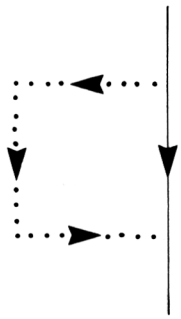


Figure 4.3

Une extension de l'idée de sauter quelques lignes de BASIC est de choisir entre deux listes différentes d'instructions. Dans ce cas il est plus facile d'en comprendre le principe après avoir vu la forme du flux de contrôle (Voir Figure 4.4).

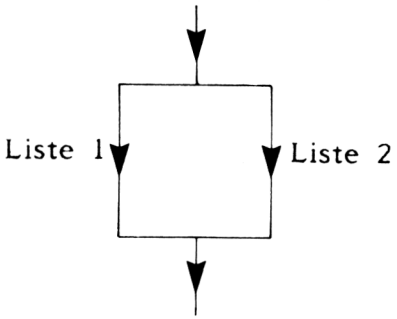


Figure 4.4

La liste à exécuter dépend de la condition utilisée dans la relation IF. La raison pour laquelle il est utile de regarder la Figure 4.4 avant d'examiner un exemple de l'ordre IF choisissant parmi deux listes d'instructions c'est qu'il n'est pas évident de voir la séparation en deux parties dans un programme BASIC. Considérez le programme suivant :

```
10 INPUT a
20 IF a<0 THEN GOTO 60
30 PRINT "a est positif"
40 LET b=a
50 GOTO 80
60 PRINT "a est negatif"
70 LET b=-a
80 PRINT a,b
```

Les lignes 30, 40, 50 ou bien 60 et 70 seront exécutées suivant la valeur de "a". Le point de division dans le diagramme correspond à la relation IF elle-même (ligne 20). Le point de jonction est la ligne 80 parce que c'est la première instruction qui sera exécutée quelle que soit la liste sélectionnée. Si vous essayez de superposer la Figure 4.4 sur le programme, vous verrez que, bien qu'elle représente ce qui se passe, il est difficile de les faire coïncider. La raison de ceci est qu'il est impossible en BASIC d'écrire les deux listes l'une à côté de l'autre. Ainsi la Figure 4.5 illustre mieux le diagramme du flux de contrôle.

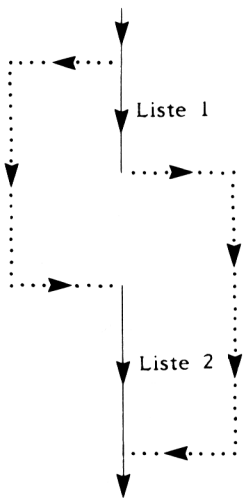


Figure 4.5

Si vous regardez la Figure 4.5 vous verrez que c'est une forme dérivée de la Figure 4.4. L'utilisation de IF pour choisir entre deux alternatives est plus facile à comprendre sur la Figure 4.4, mais la Figure 4.5 correspond davantage à la réalité.

Nous avons déjà rencontré la seule autre utilisation importante de la relation IF dans l'arrêt d'une boucle. Le diagramme du flux de contrôle dans ce cas peut être visualisé tel qu'en Figure 4.6.

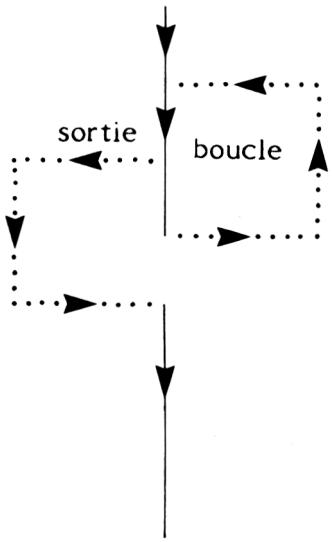


Figure 4.6

Vous devriez être en mesure de reconnaître la forme familière d'une boucle infinie. La partie qui sort de la boucle et revient au flux de contrôle normal correspond à un GOTO qui serait exécuté lorsque la condition de la relation IF est vraie. Notez que le point à partir duquel la relation IF fait une rupture de la boucle peut être placé n'importe où, de la première à la dernière instruction. Par exemple :

```
10 LET a=0
20 LET a=a+1
30 IF a=20 THEN GOTO 60
40 PRINT a
50 GOTO 20
60 PRINT "termine"
```

a son point de sortie dans le milieu et

```
10 LET a=0
20 LET a=a+1
30 PRINT a
40 IF a=20 THEN GOTO 60
50 GOTO 20
60 PRINT "termine"
```

a son point de sortie à la fin de la boucle.

En général, bien qu'il soit possible de placer le point de sortie d'une boucle n'importe où, il est préférable de la placer tout à fait au début, ou tout à fait à la fin. (Techniquement, une boucle qui a seulement un point de sortie placé au début est appelée boucle tant que et une boucle qui a seulement un point de sortie à la fin est appelée boucle jusqu'à. Toutefois ces noms proviennent d'autres langages ordinateur et ne sont pas importants dans la programmation en BASIC.)

Une boucle qui a son point de sortie à la fin peut être simplifiée en écrivant la condition d'une autre façon, par exemple :

```
10 LET a=0
20 LET a=a+1
30 PRINT a
40 IF a=20 THEN GOTO 60
50 GOTO 20
60 PRINT "termine"
```

peut s'écrire

```
10 LET a=0
20 LET a=a+1
30 PRINT a
40 IF a<>20 THEN GOTO 20
50 PRINT "termine"
```

Dans le premier exemple, la boucle se termine lorsque 'a' est égal à 20, sinon la ligne 50 contenant GOTO fait continuer la boucle. Dans le second exemple, GOTO a été éliminé en modifiant la condition de telle sorte que la boucle continue si 'a' n'est pas égal à 20. Après avoir pratiqué ce type d'instructions quelques fois, vous choisirez la méthode la plus rapide et la plus claire sans même y penser, mais il vaut mieux se rappeler à l'occasion, de quoi cela provient.

Comme évoqué au début de ce paragraphe, il y a plusieurs manières d'utiliser la relation IF pour modifier la séquence d'exécution d'un programme et vous ne devez pas vous croire limité à celles qui ont été présentées ici. Toutefois, il vaut mieux ne pas trop en essayer, car plus vous utiliserez la relation IF de manières différentes, et plus vos programmes seront difficiles à comprendre. La règle essentielle est de rendre le diagramme du flux de contrôle aussi simple que possible. Ainsi vos programmes seront faciles à lire, à comprendre et à mettre au point.

L'INSTRUCTION FOR

Mis à part la position du point de sortie, les boucles se distinguent de deux autres manières. Toutes les boucles continuent jusqu'à ce qu'une condition soit satisfaite, mais dans la plupart des cas, ceci revient à exécuter la boucle un certain nombre de fois. Par exemple, si vous vouliez afficher le mot "Bonjour" cinq fois sur l'écran, vous devriez faire :

```
10 LET a=1
```

```
20 PRINT "Bonjour"
30 LET a=a+1
40 IF a<=5 THEN GOTO 20
```

Cependant, c'est pour ce cas fréquent que le BASIC offre deux ordres supplémentaires - FOR et NEXT - pour permettre de répéter plus facilement une liste d'instructions. Le programme qui affiche "Bonjour" cinq fois sur l'écran peut s'écrire en utilisant FOR et NEXT de la façon suivante :

```
10 FOR a=1 TO 5
20 PRINT "Bonjour"
30 NEXT a
```

La signification de FOR et NEXT dans le programme devrait être claire. La variable 'a' est utilisée pour compter le nombre de fois que la boucle doit être exécutée comme dans l'exemple précédent. La différence est qu'une partie est réalisée automatiquement. L'instruction FOR donne tout d'abord à 'a' la valeur 1. A chaque fois que l'instruction NEXT est rencontrée, la valeur de 'a' est augmentée de 1. Aussi longtemps que sa valeur n'excède pas 5, le contrôle est transféré à nouveau à la ligne 20 (ceci équivaut à un GOTO 20 implicite). Le résultat est que l'ordre PRINT de la ligne 20 est exécuté cinq fois avant que le contrôle ne passe à l'instruction suivant le NEXT.

La forme générale de la boucle FOR...NEXT est :

```
FOR<variable indice>=<valeur de depart>TO<valeur finale>
     .   .   .   .
     .   .   .   .
NEXT<variable indice>
```

La variable indice est initialisée à la valeur de départ. A chaque fois que NEXT est exécuté, la variable indice est incrémentée de 1 et, tant que sa valeur n'excède pas la valeur finale, le contrôle est transféré à l'instruction suivant le FOR. La seule restriction est que le nom de la

variable indice ne doit être composé que d'une seule lettre - ainsi <variable indice> doit être une des lettres de a à z. Pour être certain que vous comprenez exactement ce que peut faire une boucle FOR, essayez les exemples suivants :

```
10 FOR i=1 TO 10
20 PRINT i
30 NEXT i

10 FOR z=100 TO 110
20 PRINT z
30 NEXT z
```

En règle générale, les valeurs de départ et valeurs de fin peuvent être des expressions arithmétiques complètes, mais il est important de réaliser qu'elles ne seront évaluées qu'une seule fois, au départ de la boucle. Ceci deviendra évident en pensant que l'ordre FOR n'est exécuté qu'une seule fois en début de boucle. La valeur de <variable indice> peut être utilisée dans des expressions arithmétiques pendant l'exécution de la boucle, mais sa valeur ne doit en aucun cas être changée. En d'autres termes, vous pouvez utiliser la variable indice dans un ordre LET, à droite du signe =, mais surtout pas à sa gauche. Le petit programme suivant affichera une table de multiplication :

```
10 INPUT "quelle table? ";t
20 INPUT "depart a? ";d
30 INPUT "fin a? ";f
40 FOR i=d TO f
50 PRINT i;"x";t;"=";i*t
60 NEXT i
```

Remarquez que les deux valeurs de début et de fin de la boucle FOR sont des expressions arithmétiques qui se ramènent en fait à des variables simples ! Notez aussi l'utilisation de la variable indice 'i' en ligne 50.

Dans la plupart des cas, on peut se servir de la forme simple de la boucle FOR, cependant on utilise quelquefois une forme plus évoluée qu'il vaut mieux connaître. Dans le cas simple de la boucle FOR, on ajoute 1 à la variable indexée à chaque fois que l'on passe dans la boucle. Ceci est intéressant si vous voulez vous servir de la variable indice pour compter le nombre de fois que la boucle est exécutée. Toutefois, on rencontre des cas où il est nécessaire que la variable indice soit incrémentée ou décrémentée de plus de 1 à chaque passage dans la boucle. Ceci est réalisé par l'adjonction de l'ordre BASIC, STEP (pas) dont la forme générale est la suivante :

```
FOR <variable indice>=<valeur de debut>
    TO <valeur de fin> STEP
    <increment>
```

L'incrément précisé après STEP est la valeur qui est ajoutée à la variable indice à chaque passage dans la boucle. Ainsi la boucle simple FOR est équivalente à STEP 1. La seule chose à laquelle vous devez faire attention en utilisant FOR STEP est d'être certain de savoir quand la boucle s'arrêtera. La règle est que la boucle se termine quand la valeur de la variable indice excède la valeur de fin. Essayez les exemples suivants :

```
10 FOR a=4 TO 10 STEP.01
20 PRINT a
30 NEXT a
```

ou

```
10 FOR a=1 TO 100 STEP 25
20 PRINT a
30 NEXT a
```

La valeur de l'incrément suivant STEP peut être négative, dans ce cas il est préférable de l'appeler décrément. Par exemple :

```
10 FOR a=100 TO 0 STEP -15
20 PRINT a
30 NEXT a
```

Il se peut que vous ayiez quelques difficultés à comprendre comment se termine ce type de boucle. Cependant, la règle est presque la même que dans le cas d'incrément positif. A chaque passage dans la boucle, la valeur de la variable indice est décrémentée et la boucle se termine lorsque cette valeur devient plus petite que la valeur de fin.

De même que l'incrément peut être négatif, les valeurs de début ou de fin peuvent l'être également, et cela peut amener une confusion. Considérez ce programme :

```
10 FOR a=-100 TO 50 STEP 10
20 PRINT a
30 NEXT a
```

A première vue, cela peut vous sembler n'avoir aucun sens. Cependant, si vous gardez la tête froide vous devez vous rendre compte que les mêmes règles s'appliquent. La valeur de l'incrément est ajoutée à la variable indice à chaque passage dans la boucle jusqu'à ce que sa valeur excède la valeur de fin si l'incrément est positif, ou soit inférieure à la valeur de fin si l'incrément est négatif.

UTILISATION DES BOUCLES FOR

Il y a une ou deux règles qui régissent l'utilisation des boucles FOR. Comme vous pouvez utiliser n'importe quelle instruction BASIC valide, dans une boucle FOR, il est possible d'utiliser GOTO ou IF pour sortir d'une boucle FOR avant qu'elle ne soit terminée (c'est-à-dire avant que la variable indice ait atteint la valeur de fin). Ceci est parfaitement autorisé par le BASIC du Spectrum mais plusieurs autres versions du BASIC n'admettent pas que vous quittiez une boucle FOR sans qu'elle n'ait été achevée. A ce

sujet, il vaut donc mieux prendre l'habitude d'utiliser les boucles FOR selon la méthode générale.

Il arrive souvent qu'une combinaison de boucles FOR soit utilisée. Ceci est facile à comprendre si vous avez saisi la logique avec laquelle travaillent les boucles. Exemple :

```
10 FOR a=1 TO 10
20 FOR b=1 TO 10
30 PRINT a,b
40 NEXT b
50 NEXT a
```

est correct, parce que la boucle FOR formée par les lignes 20, 30 et 40 est complètement contenue à l'intérieur de la boucle FOR débutant à la ligne 10 et se terminant à la ligne 50. Ceci signifie que la boucle intérieure est exécutée à chaque passage de la boucle extérieure. Il est tout aussi facile de commettre une erreur et de se retrouver avec :

```
10 FOR a=1 TO 10
20 FOR b=1 TO 10
30 PRINT a,b
40 NEXT a
50 NEXT b
```

qui provoquera un message d'erreur.

IF...THEN et ":"

Il existe une version encore plus générale de l'instruction IF, que celle examinée jusque-là. De la même manière qu'il est possible de faire suivre THEN par l'ordre BASIC GOTO, vous pouvez utiliser n'importe quel ordre BASIC valide. Exemple :

```
10 INPUT a
20 IF a<0 THEN PRINT "a est negatif"
```

```
30 IF a=0 THEN PRINT "a egal zero"
40 IF a>0 THEN PRINT "a est positif"
50 GOTO 10
```

Le meilleur moyen de comprendre cela est d'imaginer que c'est une sorte de version simplifiée du IF utilisée pour sauter une instruction. Seulement dans ce cas, l'instruction n'est exécutée que si la condition est vraie. Il n'y a pas grand-chose à ajouter à cette description, excepté le fait que vous pouvez faire suivre THEN par n'importe quel ordre BASIC valide, y compris un autre IF.

Cette version étendue de l'instruction IF semble très utile au premier abord, mais vous constaterez rapidement qu'il est très fréquent d'avoir besoin de plus d'une instruction pour faire suite à une relation conditionnelle. En clair, ce qui est nécessaire est le moyen d'écrire une liste d'instructions suivant THEN. Le BASIC Spectrum donne cette possibilité, et il est important de savoir que ce n'est pas le cas de tous les autres BASIC. Vous pouvez regrouper plusieurs instructions BASIC sur une seule ligne, chacune séparée par ":" et elles seront exécutées de la gauche vers la droite. Exemple :

```
10 PRINT "premier":LET a=1:PRINT "deuxieme ";a
```

sera exécuté comme une simple ligne de BASIC de la gauche vers la droite, ce qui est équivalent à :

```
10 PRINT "premier"
20 LET a=1
30 PRINT "deuxieme ";a
```

Cette utilisation de ":" pour grouper ensemble plusieurs instructions BASIC étend effectivement la règle du flux de contrôle. Maintenant, au lieu d'exécuter les instructions

dans l'ordre croissant des numéros de ligne, celles qui sont regroupées ensemble sur une même ligne sont traitées de gauche à droite. Ceci est tout à fait comparable à notre manière de lire (c'est-à-dire de gauche à droite et de haut en bas). Bien que cette extension puisse vous faciliter la tâche, il faut se rappeler que, les autres versions de BASIC n'ayant pas toutes cette facilité, en abuser vous rendrait très dépendant du BASIC Spectrum.

Comme mentionné plus haut, l'endroit où il est intéressant de pouvoir mettre plus d'une instruction sur une même ligne est après THEN. Par exemple, à la fin de la plupart des programmes de jeux, il est habituel de demander si le joueur désire rejouer. Dans l'exemple suivant de routine qui pourrait être ajoutée à la fin d'un programme, la réponse est attendue sous la forme de 1 pour oui ou 0 pour non. Toute autre réponse sera considérée comme une erreur et provoquera l'affichage d'un message prévu à cet effet, et la question sera posée à nouveau.

```
110 INPUT "desirez-vous un autre jeu
    (oui=1,non=0) ";a
120 IF a=1 THEN GOTO xxx
130 IF a<>0 THEN PRINT "vous devez repondre
    1 ou 0":GOTO 110
140 PRINT "au revoir!!"
```

Remarquez l'utilisation de ":" en ligne 130 pour regrouper plusieurs instructions. Evidemment, les xxx de la ligne 120 devraient être remplacés par le numéro de ligne du début de programme.

Il existe une instruction BASIC simple dont nous n'avons pas encore parlé qui est très utile lorsqu'elle est utilisée avec l'ordre IF. Supposez qu'en sortie d'une condition, vous vouliez que le programme s'arrête ; pour le moment le seul moyen que nous connaissions est de mettre un GOTO qui transfère le contrôle à la fin du programme. L'ordre BASIC STOP peut être utilisé n'importe où dans un programme, pour redonner le contrôle à l'utilisateur.

Exemple :

```
10 PRINT "voulez-vous continuer (O=1,N=0)"
20 INPUT a
30 IF a=0 THEN PRINT "au revoir":STOP
40 IF a<>1 THEN PRINT "je ne comprend pas,
   vous devez repondre 1 ou 0":
   GOTO 10
50 PRINT "je continue, Maitre"
60 GOTO 10
```

Remarquez l'utilisation de ":" pour grouper plusieurs instructions en lignes 30 et 40 et l'utilisation de STOP en ligne 30.

A titre d'exemple supplémentaire des instructions IF et FOR, considérez le probleme du calcul de la distance d'arrêt traité au Chapitre 3 comme un jeu. Pour une gamme de vitesses, on demande au joueur les distances de freinage et d'arrêt total en mètres.

```
10 PRINT "distance d'arret"
20 LET marque=0
30 FOR s=10 TO 80 STEP 10
40 PRINT "quelle est la distance de
   ralentissement a ";s;" Km/h?"
50 INPUT t
60 IF t=s THEN LET marque=marque+1
70 PRINT "quelle est la distance de freinage a ";
   s;" Km/h?"
80 INPUT b
90 IF b=s*s/20 THEN LET marque=marque+1
100 PRINT "quelle est la distance
    totale d'arret ?"
```

```
110 INPUT d
120 IF d=s*s/20+s THEN LET marque=marque+1
130 NEXT s
140 PRINT
150 PRINT "votre score est";marque
160 PRINT "pour un maxi de 24"
```

RESUME DU FLUX DE CONTROLE

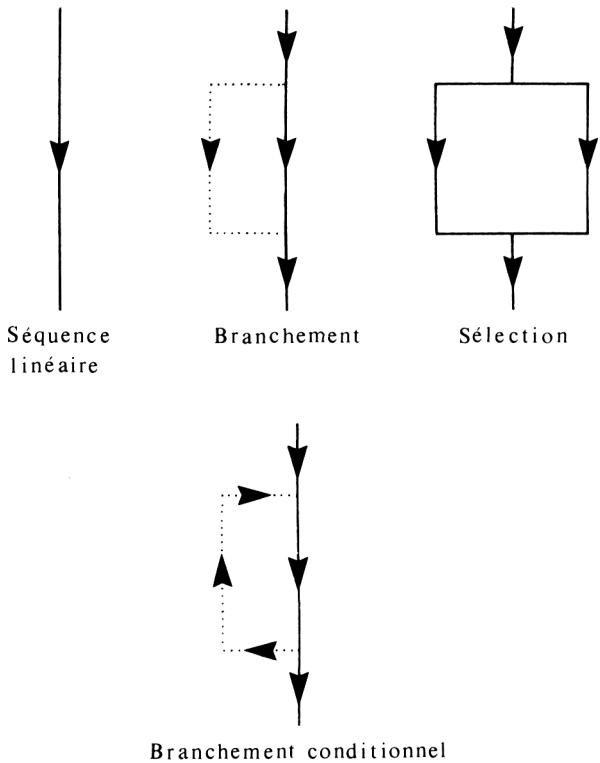


Figure 4.7 : Diagramme des flux de contrôle

Il est difficile de savoir exactement ce qui se passe dans la tête d'un programmeur expérimenté. Quoi qu'il en soit, c'est la manière de procéder qui distingue un débutant d'un expert. En conclusion de ce que nous avons vu dans ce chapitre, nous pouvons être sûrs que, quelle que soit la manière de programmer, le diagramme du flux de contrôle se ramène toujours à l'une des formes standard que nous avons étudiées et résumées en Figure 4.7.

5

MANIPULATION DE TEXTES ET DE NOMBRES

Jusqu'à présent, les programmes que nous avons écrits n'utilisaient que des nombres. Si tout ce que les ordinateurs pouvaient faire se limitait à cela, ils ne seraient pas très différents des calculatrices de poches ! Dans ce chapitre nous verrons le Spectrum peut manipuler des caractères et des textes aussi facilement que les chiffres et les nombres. La dernière partie du chapitre introduit d'autres moyens d'étendre la manipulation des données : tableaux et stockage sur cassettes.

CHAINES

Dans le Chapitre 3, la notion de chaîne (collection de lettres entre guillemets) a été introduite comme moyen d'afficher des messages et de poser des questions. En fait,

ce que nous avons appelé une chaîne est en réalité une constante chaîne. L'emploi du mot constante devrait vous faire soupçonner l'existence de variables chaînes. Une variable chaîne est semblable à une variable numérique qui est une zone de mémoire identifiée par un nom, dans laquelle il est possible de stocker une information. Dans ce cas, l'information est une suite de caractères au lieu de chiffres. Pour baptiser les variables chaînes, les règles sont différentes de celles utilisées pour les variables numériques. Un nom de variable chaîne est composé d'une lettre suivie du signe $. Le signe $ est utilisé pour faire la distinction entre les variables numériques et les variables chaînes ("b" est une variable numérique alors que "b$" est une variable chaîne donc différente). L'instruction LET peut être utilisée pour stocker une constante chaîne dans une variable chaîne, et l'instruction PRINT peut être employée pour afficher son contenu.

```
10 LET a$="ceci est une chaine"
20 PRINT a$
```

En fait, une variable chaîne peut être utilisée partout où une variable simple aurait pu l'être pour autant que cela ait un sens. Par exemple, vous pouvez utiliser INPUT a$ pour stocker une chaîne entrée à partir du clavier pendant qu'un programme s'exécute, mais LET total=3+a$ n'a évidemment pas de sens (vous ne pouvez pas ajouter une chaîne à un nombre). Notez en particulier la différence entre :

```
10 LET a$="1"
```

et

```
20 LET a$=1
```

La ligne 10 est correcte car le 1 est placé entre guillemets et de ce fait se trouve être une variable chaîne, mais 20 provoquera un message d'erreur parce que a$ est considéré comme une chaîne variable alors que 1 est ici un chiffre.

L'introduction des variables chaînes est intéressante car elle ouvre au Spectrum la possibilité de manipuler des textes et même de dialoguer. Cependant, pour l'instant, le seul type de programme que nous puissions écrire est :

```
10 INPUT "Quel est votre nom? ";n$
20 PRINT "Hello ";n$;", je suis votre
   ordinateur Spectrum"
```

qui est correct pour s'initier, mais qui aboutit à un dialogue pas très naturel tel que :

```
Quel est votre nom? Alfred Dupont
Hello Alfred Dupont, je suis votre ordinateur Spectrum
```

L'ennui est que nous pouvons entrer (INPUT), afficher (PRINT) et stocker des chaînes, mais nous n'avons pas le moyen de les modifier. C'est un peu comme si nous avions à notre disposition INPUT, PRINT, et la mémorisation pour les nombres, mais aucun moyen de faire des calculs. Il est évident que cela limiterait les programmes que nous pourrions écrire ! La solution existe sous forme d'une arithmétique des chaînes. De même que nous disposons des expressions arithmétiques, nous pouvons utiliser des expressions de chaînes.

EXPRESSIONS DE CHAINES

Avant d'introduire les moyens dont dispose le Spectrum pour manipuler les chaînes, il est plus judicieux de considérer les types de traitement que vous aimeriez faire, et de voir si au stade où nous sommes, le Spectrum en est capable ou non. Si dans un programme, on a placé un prénom dans f$ et un nom dans l$, il serait commode de pouvoir les réunir ensemble dans une chaîne plus longue. La réunion de chaînes s'appelle la concaténation. Une autre chose intéressante serait la possibilité d'extraire une partie de

chaîne ; par exemple, si vous vouliez extraire un nom à partir d'une chaîne composée d'une initiale et d'un nom, tel que récupérer "Dupont" dans une chaîne contenant "P. Dupont".

Une chaîne qui est une partie d'une autre chaîne s'appelle une sous-chaîne. "Dupont" est une sous-chaîne de "P. Dupont". Une autre commodité serait de pouvoir remplacer une sous-chaîne par une autre. Ainsi, si nous voulions essayer de garder secret "P. Dupont", nous pourrions remplacer le nom par des astérisques, tel que "P.******". Finalement, il serait plus avantageux de pouvoir tester la présence d'une sous-chaîne particulière à l'intérieur d'une chaîne, pour savoir si la sous-chaîne "Dupont" figure dans la chaîne n$, par exemple. En résumé, ce que nous aimerions trouver comme opérations sur les chaînes sont : la concaténation, l'extraction, la substitution et la recherche de sous-chaînes.

Notre première requête, la concaténation, est immédiatement satisfaite par le signe + de concaténation du Spectrum. Si a$ contient la chaîne "abcd" et b$ contient "efgh", après :

```
LET c$=a$+b$
```

c$ contiendra "abcdefgh". Notez que nous avons maintenant à notre disposition deux utilisations possibles du signe + comme signe d'addition et comme signe de concaténation. Vous pouvez utiliser le signe + plusieurs fois dans une expression de chaîne :

```
LET c$="Mr"+f$+" "+n$
```

mettra en relation les quatre chaînes évoquées, et si f$ contient le prénom "Patrick", et n$ le nom "Dupont", c$ contiendra donc "Mr Patrick Dupont". Remarquez l'utilisation de l'espace entre les chaînes f$ et n$ pour éviter d'obtenir "Mr PatrickDupont".

L'extraction ou l'échange de sous-chaîne sont obtenus en utilisant une nouvelle opération simple, le découpage. Plusieurs autres versions du BASIC utilisent des méthodes plus compliquées que le découpage de chaîne du Spectrum. Ce dernier offre davantage de facilité par rapport à d'autres BASIC, en ce qui concerne la manipulation des chaînes. Un découpeur de chaîne spécifie une sous-chaîne en donnant la position de la première et de la dernière lettre de la chaîne. Par exemple :

```
PRINT "12345678" (3 TO 6)
```

affiche la sous-chaîne "3456" débutant au 3ème caractère et se terminant au 6ème. Souvenez-vous que TO doit être entré en actionnant une simple touche (la touche F). Taper les lettres T et O serait incorrect ! La forme générale d'une instruction de découpage est :

<expression de chaîne> <expression1 arithmétique> TO <expression2 arithmétique>

La sous-chaîne spécifiée débute au caractère indiqué par <expression1> et finit au caractère donné par <expression2>, dans la chaîne <expression de chaîne>. Cette définition peut sembler compliquée, mais de fait elle va beaucoup plus loin que la plupart des formes de découpage des autres BASIC. Le plus souvent, expression de chaîne est une constante ou bien une variable chaîne simple et les expressions arithmétiques sont des constantes ou des variables simples. Par exemple :

```
"abcd" (2 TO 3)
```

ou

```
"abcd" (compte TO 3)
```

A présent que vous avez pris conscience qu'un des principes les plus puissants du BASIC consiste à pouvoir utiliser

indifféremment des expressions ou des variables, le découpage de chaîne n'y fait pas exception et ainsi vous pouvez écrire des instructions telles que :

("abcd"+"efgh")(debut TO debut+3)

dans laquelle la première expression concatène les deux chaînes "abcd" et "efgh" en une simple chaîne "abcdefgh" dont on extrait 4 lettres à partir de la position indiquée par le contenu de la variable simple "début". Ecrire des expressions de chaînes compliquées ne devrait pas vous effrayer plus que les expressions arithmétiques.

Il y a un certain nombre de cas particuliers dans la notation de découpage, qu'il vaut mieux connaître. Si vous omettez <expression1 arithmétique>, la sous-chaîne partira du début de la chaîne et si vous ignorez <expression2 arithmétique>, la fin de la chaîne sera prise en compte. Ainsi :

"123456"(TO 5) est équivalent à "123456"(1 TO 5) donnant "12345"
"123456"(3 TO) est équivalent à "123456"(1 TO 6) donnant "3456"
"123456"(TO) est équivalent à "123456"(1 TO 6) donnant "123456"

Il y a en plus une abréviation très utilisée pour extraire un seul caractère de n'importe quelle position d'une chaîne. Au lieu d'avoir à écrire (n TO n) pour extraire le caractère situé à la position n dans une chaîne, vous pouvez écrire simplement (n) tel que :

"123456"(3) est équivalent à "123456"(3 TO 3) qui donne "3"

Il est possible de commettre des erreurs en utilisant le découpage. Par exemple, si vous spécifiez une position inexistante, vous obtiendrez un message d'erreur disant SUBSCRIPT WRONG (position erronée) ou bien INTEGER OUT OF

RANGE (nombre entier hors limites). Par exemple, "123456" (n TO 7) ou "123456"(-1 TO 3) entraînerait des messages d'erreur. Cependant si vous donnez une position finale inférieure à la position de départ "123456"(3 TO 1), vous serez surpris de ne voir apparaître aucun message d'erreur. De ce fait, la réponse est une forme toute particulière de chaîne : la chaîne nulle. La chaîne nulle ne contient aucun caractère et joue un rôle très semblable à celui de zéro dans les expressions arithmétiques. La chaîne constante correspondant à la chaîne nulle s'écrit "" (une paire de guillemets accolés). Notez la différence entre "" et " ". La première chaîne est la chaîne nulle et ne contient donc aucun caractère, alors que la seconde est une chaîne d'un caractère : un blanc ou espace. Du point de vue de l'ordinateur, le caractère blanc est exactement comme une lettre de l'alphabet, il occupe une position pour les impressions, et une place en mémoire pour le stocker. Si vous affichez une chaîne nulle, cela n'aura aucun effet. Les instructions PRINT a;b et PRINT a;"";b produiront le même résultat.

Comme exemple d'utilisation de découpage de chaîne, examinez le problème consistant à afficher le nom d'un mois repéré par son numéro (vous afficheriez Dec pour 12 et Mai pour 5, etc.). Essayez le petit programme suivant :

```
10 LET a$="JanFevMarAvrMaiJunJuiAouSepOctNovDec"
20 INPUT "Numero du mois? ";num
30 PRINT "le mois ";num;" est ";a$(1+3*(num-1)
   TO 3*num)
40 GOTO 20
```

Si vous donnez un nombre compris entre 1 et 12, le programme affichera correctement l'abréviation du mois correspondant. Ceci est obtenu par extraction d'un nom de trois lettres à partir de la chaîne a$. Le meilleur moyen de comprendre le découpeur de la ligne 30 est de le faire fonctionner à la main en prenant quelques valeurs de 'num'. Si 'num' vaut 4,

ceci entraîne que (1+3*(num-1))=10 et 3*num=12, (10 TO 12) spécifie les 3 lettres "Avr".

Le découpage n'est pas seulement utilisé pour extraire des sous-chaînes. Vous pouvez utiliser la notation standard du découpage en partie gauche du signe = d'un ordre LET pour définir la sous-chaîne à modifier. Par exemple :

```
10 LET a$="123456"
20 LET a$(3 TO 4)="ABCDEFG"
30 PRINT a$
```

affichera "12AB56" sur l'écran. Cette caractéristique est une extension très logique de l'utilisation habituelle de LET, pour stocker une chaîne dans une variable chaîne. Le découpage restreint simplement le nombre des caractères à modifier. Si la chaîne située en partie droite du signe = d'un LET est plus grande que la sous-chaîne spécifiée, les caractères de débordement ne seront pas pris en compte, et si elle est plus courte, elle se trouvera complétée par des blancs. Exemple :

```
10 LET a$="123456"
20 LET a$(2 TO 5)="ABC"
30 PRINT a$
```

affichera la chaîne "1ABC 6". (Remarquez le blanc suivant le caractère C.)

La seule chose qu'il nous reste à voir dans notre liste de manipulations de chaînes est de tester la présence d'une sous-chaîne particulière dans une chaîne. Le BASIC du Spectrum ne dispose d'aucune méthode directe pour cela, par contre l'extension des expressions conditionnelles (voir Chapitre 4) aux chaînes permet d'y arriver. Vous pouvez utiliser toutes les relations introduites dans ce chapitre, avec les chaînes. La signification de = et <> est facile à comprendre. Deux chaînes sont égales si elles ont la même longueur, les mêmes caractères et dans le même ordre, sinon elles sont inégales. Cependant, quelles sont les signi-

fications des opérateurs <, >, <= et >= appliqués aux chaînes ? La réponse à cette question diffère suivant les BASIC. Néanmoins, dans le cas du Spectrum ils sont définis tels que a$<b$ est vrai si la chaîne a$ apparaît avant la chaîne b$ par ordre alphabétique. En fait, nous sommes tellement habitués aux listes classées par ordre alphabétique, que nous avons tendance à en oublier le principe. Si les deux chaînes à comparer contiennent une seule lettre, alors a$<b$ si la lettre contenue dans a$ est placée dans l'alphabet avant la lettre contenue dans b$. Par exemple, "a"<"b" est vrai mais "d"<"b" est faux. La comparaison de chaînes composées d'un unique caractère ne porte pas nécessairement que sur des lettres, par exemple que faisons-nous de "*"<"$" ? Dans le cas des lettres, l'alphabet étant ordonné, il peut servir de référence. Ainsi ce dont nous avons besoin est une extension ordonnée de tous les caractères que le Spectrum utilise. En d'autres termes, nous avons besoin d'un super alphabet ! Et ceci existe sur le Spectrum. Si vous regardez l'Appendice A de son manuel d'utilisation vous y trouverez une liste complète et ordonnée de tous les caractères et ceci est utilisé comme super alphabet pour décider si a$<b$. Si vous ne disposez pas du Manuel du Spectrum ou simplement si cela vous intéresse, vous pouvez afficher tous les caractères dans l'ordre en utilisant le programme suivant :

```
10 FOR i=34 TO 255
20 PRINT "caractere ";i;" = ";CHR$(i)
30 NEXT i
```

La ligne 10 précise 34 comme point de départ, car les caractères précédents ne sont pas imprimables. Ne vous tracassez pas au sujet de CHR$ en ligne 20, cette fonction sera traitée plus tard. Si vous lancez ce programme, vous serez surpris de voir apparaître des mots tels que PRINT et FOR à la place de caractères ! Ceci est dû au fait que le Spectrum considère comme caractère tout ce qui peut être

entré directement au clavier à partir d'une simple touche, même si s'agit d'un mot ! Revenons à la question de savoir si oui ou non "*"<"$" est vrai, "*" est le 42ème caractère et "$" est le 36ème, venant donc avant "*" ; en conséquence "*"<"$" est faux. Vous pouvez tester de la même façon n'importe quelle relation.

Si les deux chaînes contiennent plus d'un caractère, elles sont comparées caractère par caractère jusqu'à ce que la première différence soit rencontrée. La relation entre deux chaînes est ainsi décidée sur la base de deux caractères. Par exemple, ABCD<AZCD est vrai car la première paire de lettres différentes est B et Z et que B<Z est vrai. Si une chaîne est identique à une partie d'une autre chaîne, la comparaison se fera par rapport à la longueur. Exemple : ABCD<ABCDEF est vrai parce qu'il n'y a aucune paire de lettres qui soient différentes, et que ABCD est plus court que ABCDEF.

La notion de longueur de chaîne est relativement importante, et le Spectrum offre un moyen très simple pour trouver la longueur de n'importe quelle chaîne. Essayez le programme suivant :

```
10 INPUT a$
20 PRINT LEN(a$)
30 GOTO 10
```

vous verrez qu'il affiche le nombre de caractères que vous avez tapés en réponse à INPUT. En général :

LEN (<expression de chaine>)

donnera le nombre de caractères contenus dans <expression de chaîne>. (L'utilisation de LEN sera traitée plus en détail au Chapitre 6.)

TABLEAUX

Les chaînes et les nombres sont les deux seuls types de données que le Spectrum sache manipuler et cela est suffisant dans la plupart des cas. Cependant, le Spectrum offre la possibilité de manipuler des données par un moyen plus sophistiqué, le tableau.

Considérez le problème suivant : lire 5 nombres et les afficher à l'écran en ordre inverse. Actuellement, la seule méthode que nous utiliserions est:

```
10 INPUT a1,a2,a3,a4,a5
20 PRINT a5,a4,a3,a2,a1
```

ce qui ne serait pas trop mauvais pour 5 nombres, mais imaginez à quoi ressemblerait le programme si le problème était d'inverser 100 nombres !

Ce dont nous avons besoin est de pouvoir nous référer à une variable comme a(i) dans laquelle i peut prendre les valeurs de 1 à 5 dans une boucle. Ainsi nous pourrions écrire :

```
10 FOR i=1 TO 5
30 INPUT a(i)
40 NEXT i
50 FOR i=5 TO 1 STEP -1
60 PRINT a(i)
70 NEXT i
```

C'est exactement ce que BASIC nous permet de faire. La collection de variables a(1) à a(5) est appelée le tableau et une variable particulière a(i) est appelée un élément du tableau (Voir Figure 5.1).

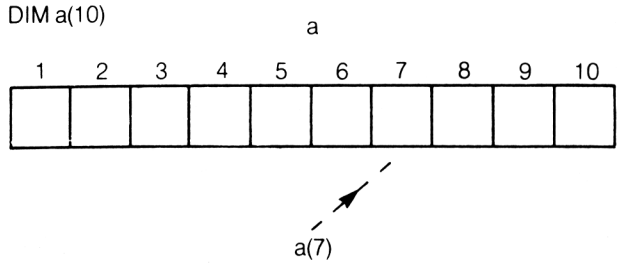


Figure 5.1 : Tableau à une dimension

La seule complication est qu'avant d'utiliser un tableau, vous devez dire au Spectrum de combien d'éléments il est composé. Ceci est réalisé par l'ordre DIM :

```
10 DIM a(5)
```

qui, ajouté au programme précédent, lui permettrait de fonctionner ! Si vous définissez une variable comme ayant seulement 5 éléments et que vous essayez d'utiliser a(6), un message d'erreur sera produit vous signalant que vous essayez d'utiliser quelque chose qui n'existe pas. Il est tentant de penser que pour éviter les messages d'erreur, il suffit de définir un tableau plus grand que nécessaire, mais attention, les tableaux peuvent très rapidement utiliser toute la mémoire disponible ! De plus, il existe une autre restriction, le nom des variables indicées des tableaux qui ne peuvent être définis que par une seule lettre ! Cela signifie que vous ne pouvez définir plus de 26 tableaux (de 'a' à 'z') dans un même programme, ce qui en pratique est bien suffisant. L'utilisation de l'ordre DIM détruit tout tableau déjà existant portant le même nom et en crée un nouveau initialisé de telle sorte que chaque élément soit égal à zéro.

En plus, afin de concevoir des tableaux de plusieurs lignes de variables, vous pouvez les définir sous forme de lignes et colonnes. Par exemple :

```
DIM a(10,10)
```

définit une collection de variables organisées en 10 lignes de 10 colonnes chacune. Un élément particulier de ce tableau est identifié par a(i,j) dans lequel i désigne le numéro de ligne et j la colonne.

Ce concept de tableaux à deux dimensions peut être étendu à trois, quatre, etc., jusqu'à 255 dimensions. Il est difficile d'imaginer la structuration des variables du tableau de plus haute dimension, pourtant la méthode est la même que pour les tableaux à une ou deux dimensions. Par exemple :

```
DIM a(10,20,5)
```

est un tableau à trois dimensions duquel on peut extraire par exemple l'élément a(2,1,4). En fait, il n'est pratiquement pas fait usage de tableaux supérieurs à 2 dimensions, à cause de l'occupation mémoire qui croît très rapidement !

Vous pouvez composer des tableaux utilisant des chaînes de caractères au même titre que les nombres, ce qui vous permet de stocker et manipuler des listes de mots. Il y a cependant un petit problème du fait qu'une chaîne de caractères est déjà un tableau à une dimension. En définissant ce genre de tableau de chaîne, cela revient en fait à spécifier la longueur de cette chaîne. Exemple :

```
DIM s$(10)
```

est une variable chaîne qui peut contenir au plus 10 caractères (y compris les caractères blancs). Pour voir la différence entre la chaîne r$ et le tableau s$ essayez le programme suivant :

```
10 DIM s$(10)
20 LET s$="abc"
30 LET r$="abc"
40 PRINT s$;"x";r$;"x"
```

Vous verrez alors que s$ s'affiche sur 10 caractères indépendamment de ce que vous y avez stocké. Remarquez que dans un tableau de caractères, vous pouvez aussi bien stocker des valeurs dans plus d'un élément à la fois que dans un seul. Les lignes suivantes stockent "z" dans le cinquième élément du tableau s$.

```
50 LET s$(5)="z"
60 PRINT s$
```

Vous devez trouver une similitude entre ceci et le découpage introduit précédemment.

Il est possible de manipuler des listes de mots à l'aide de tableaux à deux dimensions. Par exemple, le programme qui inversait les nombres, peut être réutilisé pour inverser une liste de mots de la façon suivante :

```
10 DIM a$(5,10)
20 FOR i=1 TO 5
30 INPUT a$(i)
40 NEXT i
50 FOR i=5 TO 1 STEP -1
60 PRINT a$(i)
70 NEXT i
```

Notez que si vous laissez de côté le dernier indice d'un tableau à deux dimensions (ou plus), cela revient à traiter le tableau comme une collection de chaînes.

a$(5,2) représente un simple caractère
mais
a$(5) représente une chaîne de 10 caractères.

Il est possible d'utiliser la notation de découpage pour extraire des sous-chaînes :

```
a$(5,1 TO 2)
```

extrait une sous-chaîne comprenant le premier et le deuxième caractère.

Le BASIC Sinclair est très souple dans l'utilisation des chaînes et tableaux. Cela n'est peut-être pas évident immédiatement, mais une fois que vous vous serez familiarisé avec, vous trouverez cela très puissant.

JEU DE MOTS

A titre d'exercice sur l'utilisation des tableaux de caractères, écrivons un programme de jeu : le jeu du pendu. Parce que l'ordinateur ne peut pas penser à une liste de mots que vous devez deviner, il est nécessaire de demander à quelqu'un d'autre de lui entrer cette liste. Après que cette dernière ait été enregistrée, le joueur doit essayer de deviner chaque mot, lettre par lettre. Chaque lettre entrée doit être comparée à chacune des lettres composant le mot. Si la lettre testée est bonne, elle est remise à blanc dans le mot pour éviter de la tester une seconde fois. Lorsque toutes les lettres ont été devinées, le programme passe au mot suivant, et s'il n'y en a plus, passe à la fin.

Après cette description du problème, vous devriez être apte à réaliser votre propre programme du pendu, avant de regarder celui qui suit :

```
10 DIM m$(5,10)
20 FOR i=1 TO 5
30 INPUT "Mot = ";m$(i)
40 NEXT i
50 FOR i=1 TO 5
60 LET g=0
70 LET t$=m$(i)
80 LET g=g+1
90 INPUT "essai = ";a$
100 LET f=0
110 FOR j=1 TO 10
120 IF a$(1)=m$(i,j) THEN PRINT"Oui!";
    a$(1):LET m$(i,j)=" "
```

```
130 IF m$(i,j)<>" " THEN LET f=1
140 NEXT j
150 IF f=1 THEN GOTO 80
160 PRINT "Vous avez trouve en ";g;" coups !!"
170 PRINT "Le mot etait ";t$
180 NEXT i
190 PRINT "Fin du jeu"
```

La ligne 10 définit le tableau m$ de telle sorte qu'il puisse contenir 5 mots de 10 lettres. Les lignes 20 à 40 servent à entrer les mots. La tranche de 50 à 180 permet de deviner les mots et se répète 5 fois, une pour chaque mot. La variable t$ en ligne 70 sert à stocker le mot jusqu'à ce qu'il ait été deviné, ce qui permet de remettre à blanc chacune des cases correspondant à une lettre trouvée du mot. La devinette est posée en ligne 90. Chaque lettre du mot est testée par la boucle débutant en ligne 110 et se terminant en ligne 140. La variable 'f' sert à indiquer qu'il reste des lettres à deviner.

INITIALISATION DE VARIABLES - DATA ET RESTORE

Il arrive fréquemment que l'on doive stocker des caractères dans des variables ou des tableaux afin de rendre un programme utilisable. Par exemple, supposez que nous voulions imprimer le nombre de jours d'un mois donné. Nous serions conduits à constituer une table de 12 zones et y stocker la réponse pour chaque mois, telle que le nombre de jours de janvier serait dans la zone 1, février dans la zone 2, etc. Ceci est un concept très simple et très usité, mais comment initialiser chaque élément du tableau à la bonne valeur ? Vous pourriez utiliser une boucle FOR et un INPUT pour entrer les valeurs ou encore 12 LET. Pour se faciliter la vie, le BASIC offre une facilité pour stocker des données à l'intérieur d'un programme de telle sorte qu'elles puissent être transférées à n'importe quelle

variable de votre choix. La donnée est stockée dans l'instruction DATA qui se compose de ce simple mot suivi de la liste des valeurs, séparées par des virgules. Ainsi le nombre de jours des mois pourrait s'écrire :

```
10 DATA 31,28,31,30,31,30,31,31,30,31,30,31
```

Pour transférer le contenu de ces données dans d'autres variables, l'ordre READ est utilisé. Un ordre READ s'écrit suivi d'une liste de variables séparées par des virgules. A chaque fois qu'un ordre READ est rencontré, la valeur des données est transférée dans la variable correspondante de la liste et ce, une par une. La meilleure façon de se représenter cela, est de s'imaginer un pointeur positionné au départ vers la première valeur contenue dans la liste de l'ordre DATA. A chaque fois qu'un ordre READ transfère le contenu d'une valeur, le pointeur est déplacé vers la case suivante. Ainsi on peut accéder aux 12 valeurs de l'ordre DATA décrit plus haut, de la manière suivante :

```
20 DIM m(12)
30 FOR i=1 TO 12
40 READ m(i)
50 NEXT i
```

Vous pouvez utiliser autant d'ordres DATA que vous le souhaitez dans un programme, ils sont traités comme s'il s'agissait d'une même liste DATA. Ainsi lorsque le pointeur dépasse la fin d'un ordre DATA, il se positionne au début de la liste DATA suivante. S'il n'en existe pas de suivante, un message d'erreur s'affichera.

Aucune restriction n'est appliquée au type de données que l'on peut stocker dans une liste DATA (vous pouvez aussi bien utiliser des chaînes que des nombres) mais vous devez toujours prêter attention à bien faire correspondre un ordre READ avec un DATA de même type (chaînes avec chaînes, nombres avec nombres). Les chaînes doivent être délimitées

par des guillemets, autrement il ne serait pas possible de dire où se termine un élément de donnée et où commence le suivant. A titre d'illustration des chaînes dans un ordre DATA regardez ceci :

```
10 DATA "jan","fev","mar","avr","mai","jun",
   "jui","aou","sep","oct","nov","dec"
20 DIM m$(12)
30 FOR i=1 TO 12
40 READ m$(i)
50 NEXT i
```

et comparez-le à l'exemple précédent.

Parfois, et en particulier dans les jeux, il est intéressant de pouvoir modifier la position du pointeur imaginaire. Cela peut être réalisé grâce à la commande RESTORE. RESTORE suivi d'un numéro de ligne aura pour effet, au prochain READ, de récupérer la valeur contenue dans le premier élément de la liste DATA correspondant à ce numéro de ligne. (S'il n'existe pas de DATA à ce numéro de ligne, un message d'erreur sera affiché.) Si vous utilisez RESTORE sans numéro de ligne, le premier ordre DATA sera pris en compte. Essayez par exemple :

```
10 DATA 1,2,3,4
20 READ a,b,c,d
30 PRINT a,b,c,d
40 RESTORE
50 GOTO 20
```

SAUVEGARDE DES DONNEES SUR UNE CASSETTE

Nous avons déjà vu qu'il était possible de sauvegarder et de charger des programmes sur cassettes. Il semble logique d'étendre cette facilité à la sauvegarde (SAVE) et au chargement (LOAD) de données stockées dans des variables sur

ou à partir d'une bande. En fait, le Spectrum donne seulement la possibilité de sauvegarder et de charger des tableaux, mais ceci n'est pas un inconvénient ; une simple variable que vous voudriez sauvegarder devrait être au préalable transférée dans les cases libres d'un tableau et alors celui-ci pourrait être ensuite sauvegardé. La seule chose que vous ayez à faire pour sauvegarder un tableau sur bande est de procéder exactement comme si vous vouliez sauvegarder un programme, mais utilisez :

```
SAVE "nomdefichier" DATA "nomdetableau" ()
```

où "nomdefichier" est une chaîne constante ou variable, qui est le nom du fichier de données et "nomdetableau" le nom du tableau, numérique ou chaîne, devant être sauvegardé. (Remarquez les parenthèses vides.) De la même façon, pour charger un tableau précédemment sauvegardé, utilisez :

```
LOAD "nomdefichier" DATA "nomdetableau" ()
```

Ceci aura pour effet de chercher sur la bande jusqu'à ce qu'un tableau précédemment enregistré soit trouvé avec le nom de fichier correct. Tout tableau existant portant le même nom serait effacé de la mémoire et remplacé par la version sauvegardée. Vous pouvez également utiliser :

```
VERIFY "nomdefichier" DATA "nomdetableau" ()
```

pour contrôler que les données ont bien été sauvegardées sans erreur, de la même façon que pour les programmes. A titre d'exemple de sauvegarde et de chargement de données, essayez :

```
10 DIM a(10)
20 FOR i=1 TO 10
```

```
30 LET a(i)=i
40 NEXT i
50 PRINT "rembobinez la bande"
60 SAVE "test" DATA a()
70 PRINT "rembobinez la bande et pressez <play>"
80 LOAD "test" DATA a()
90 PRINT "donnees chargees OK"
```

Mis à part le fait qu'il faut se souvenir d'appuyer sur les bonnes touches du magnétophone, c'est aussi simple que cela.

6

FONCTIONS ET SOUS-PROGRAMMES

A ce stade de l'étude du BASIC, vous devez avoir constaté que le meilleur moyen de modifier des données est d'utiliser des expressions. Sans les expressions arithmétiques, conditionnelles et de chaînes, le BASIC se réduirait simplement à déplacer des valeurs d'une zone à l'autre. C'est seulement en utilisant des expressions que les valeurs peuvent être combinées et comparées afin d'obtenir de nouveaux résultats. Pour créer des expressions toujours plus puissantes, le BASIC offre une grande gamme d'opérations qui peuvent être utilisées sous forme de fonctions. Vous pouvez définir vos propres fonctions. Cette création de nouvelles opérations sera développée plus tard avec l'utilisation des ordres GOSUB et RETURN qui permettent de regrouper des instructions dans des unités fonctionnelles ou sous-programmes.

Dans la première partie de ce chapitre, nous verrons le concept général de fonction puis nous examinerons quelques-unes des fonctions les plus utiles au programmeur de Spectrum. Les paragraphes qui traitent des fonctions particulières peuvent être lus très rapidement ou même écartés jusqu'à ce que vous en ayez besoin ou que vous les rencontriez dans des exemples. Toutefois ne sautez pas le paragraphe sur les fonctions spéciales car il est particulièrement important.

LE CONCEPT DE FONCTION

Avant d'examiner comment le Spectrum manipule les fonctions, il est nécessaire de regarder les fonctions en général. Les mathématiques ont déjà dû vous familiariser avec les fonctions. SIN (x), par exemple, est une fonction. Cependant, ici le concept de fonction n'est pas réellement axé sur les mathématiques. Réduite à sa plus simple expression, une fonction est une opération sur une donnée qui produit un résultat unique. Trouver le plus grand de deux nombres est un exemple d'opération sur des données qui rend un résultat unique. Le maximum de 3 et 42 est 42, le maximum de 2 et 2 est 2 ; dans ces exemples, maximum est une fonction.

Le Spectrum n'est pas pourvu d'une telle fonction, elle n'a été utilisée ici qu'à titre d'exemple pour faciliter la compréhension, mais comme nous le verrons plus loin, il est très facile d'en programmer une.

La façon standard d'écrire une fonction implique d'écrire son nom à gauche des valeurs définies entre parenthèses. Dans le cas où il faut trouver le maximum de deux nombres, nous pouvons baptiser cette fonction "max" et écrire les deux exemples précédents ainsi :

```
max(3,42)
  et
max(2,2)
```

Suivant les conventions habituelles du BASIC, vous pouvez substituer une expression à n'importe quelle constante ou variable, ainsi vous pouvez en écrire une du genre :

```
max(compte+3,total*20)
```

Les données qui suivent le nom d'une fonction sont appelées paramètres. Il est possible d'avoir plusieurs paramètres par fonction, mais sur le Spectrum elles n'en ont qu'un seul.

Les fonctions peuvent être utilisées comme expressions au même titre que des variables ou constantes. Par exemple :

```
LET resultat=max(3.3,4.2)
```

évalue notre fonction "max" et mémorise la réponse (4.2) dans "résultat". (Rappelez-vous que le Spectrum n'a pas de fonction max, n'essayez donc pas cet exemple.) Si une fonction donnait plus d'un résultat, lequel devrait être stocké dans "résultat" ou encore lequel devrait être pris pour évaluer le reste de l'expression ? Comme nous voulons utiliser les fonctions dans des expressions, elles ne doivent retourner qu'une seule réponse. Il est possible parfois de transformer en fonction quelque chose qui n'en est pas une, en choisissant simplement une des réponses possibles. Par exemple, le Spectrum a une fonction SQR qui donne la racine carrée d'un nombre. Maintenant si vous demandez la racine carrée de quatre, la réponse est évidemment deux, car deux fois deux font quatre, mais il est facile de voir que moins deux est aussi la racine carrée de quatre. Moins par moins égale plus et donc -2*-2 font 4 (et non -4). On peut modifier le fait que l'opération d'extraction d'une racine carrée ne soit pas une fonction, en décidant arbitrairement que SQR en sera une qui ne donnera que la valeur positive.

LES FONCTIONS DU SPECTRUM

Le Spectrum possède une grande variété de fonctions et quelques-unes de celles-ci sont si spécialisées qu'il est préférable de les traiter en détail dans d'autres chapitres. Cependant il existe un noyau central de fonctions que l'on peut trouver dans n'importe quel BASIC, et celles-ci vous seront expliquées dans ce chapitre. La liste complète et commentée des fonctions du Spectrum se trouve en Appendice C du Manuel du Spectrum.

Le noyau des fonctions peut être divisé en trois groupes : les fonctions arithmétiques telles que SQR (racine carrée) et ABS (absolue), les fonctions trigonométriques telles que SIN (sinus) et COS (cosinus) et les fonctions de chaînes telles que LEN et CHR$. Il en existe d'autres plus difficiles à classer, mais dont l'une est très importante, RND. Toutes les fonctions sont entrées à l'aide d'une simple touche, et l'utilisation de parenthèses autour des paramètres est optionnelle. La chose la plus importante est d'avoir une idée des fonctions valides ; il est donc recommandé de lire la description de chacune des fonctions ci-dessous. Cependant, il est difficile d'apprécier les détails les plus subtils d'utilisation d'une fonction tant que vous ne vous en êtes pas servi ! Si vous voulez voir le résultat que produit n'importe quelle fonction, exécutez le programme suivant :

```
10 INPUT x
20 PRINT SIN x
30 GOTO 10
```

et remplacez SIN x par la fonction qui vous intéresse.

Fonctions arithmétiques

ABS - Valeur absolue d'un nombre

La valeur absolue d'un nombre est obtenue en ignorant son

signe et en le traitant comme étant positif, c'est-à-dire ABS(-2) et ABS(2) sont égales à 2.

EXP - Fonction exponentielle
EXP(x) est calculé en élevant "e" à la puissance "x" (e a la valeur 2.718281). EXP(x) est donc équivalent à e^x.

Il est difficile d'expliquer pourquoi cette fonction est si importante, mais on la retrouve partout en mathématiques (voir aussi la fonction LN). Lorsque l'on utilise EXP, il faut être conscient du fait que EXP(x) donne de très grands nombres même pour de petites valeurs de (x). Le plus grand nombre entier que le Spectrum peut traiter avec EXP(x) est 88. Des nombres plus grands provoqueront l'affichage du message number too big (nombre trop grand).

INT - Valeur entière d'un nombre
La fonction INT est probablement la plus simple et la plus utilisée de toutes les fonctions arithmétiques. A partir d'un nombre quelconque, elle retourne un nombre entier en l'arrondissant par défaut. Pour les nombres positifs, ceci revient à couper la partie fractionnaire, exemple : INT(3.21) égale 3, mais pour les nombres négatifs les choses sont un peu plus compliquées. Arrondir par défaut un nombre négatif paraît un peu bizarre, INT(-4.7) égale -5, simplement parce que -4>-5 c'est-à-dire que -5 est plus petit que -4 !

LN - Logarithme naturel d'un nombre
Le logarithme naturel d'un nombre est la puissance à laquelle il faut élever "e" pour obtenir ce nombre. La plupart des gens sont habitués à une forme légèrement différente de logarithmes. Ce qui est donné dans beaucoup de tables de logarithmes est en fait le logarithme de base 10. En d'autres termes, c'est la puissance à laquelle 10 doit être élevé pour redonner le nombre d'origine. Quelques versions de BASIC incluent les logarithmes de base 10 sous la forme d'une fonction LOG. Le Spectrum n'a pas ce type de fonction, mais ce n'est pas un problème, car vous pouvez

obtenir le LOG de base 10 en utilisant LN(x)/LN(10). Comme LN(x) est le nombre auquel il faut élever "e" pour obtenir "x", vous devriez être en mesure de voir que EXP(LN(x)) vaut x.

PI (π)
C'est une fonction très bizarre qui n'a aucun paramètre et donne toujours le même résultat (π=3.14159265) mais qui est très utile. Comme chacun le sait, la surface d'un cercle de rayon r est donnée par πr2 et ceci est traduit en BASIC par :

```
10 LET surface=PI*r*r
```

SGN - Le signe d'un nombre
Le signe d'un nombre est +1 si le nombre est positif et -1 s'il est négatif. Par exemple, SGN(-232) est -1 et SGN(3239) est 1.

SQR - La racine carrée
La racine carrée d'un nombre est le nombre qui, multiplié par lui-même, redonne le nombre d'origine, c'est-à-dire :

SQR(x)*SQR(x) égale x

Remarquez qu'un nombre négatif n'a pas de racine carrée, car si vous multipliez un nombre négatif par lui-même, vous obtiendrez un nombre positif. Si vous essayez d'extraire la racine carrée d'un nombre négatif, vous obtiendrez alors un message d'erreur très précis invalid argument (argument invalide).

Fonctions trigonométriques

Les exemples de fonctions les plus connues sont probablement les fonctions trigonométriques. La théorie de la trigonométrie dépasse le cadre de ce livre et de toute

façon la nécessité d'utiliser de telles fonctions émane toujours de problèmes très spécifiques. Il y a cependant une utilisation de fonctions trigonométriques qui est importante pour tous ceux que les graphiques intéressent, à savoir les dessins de cercles. Pour en savoir plus sur le sujet, reportez-vous au Chapitre 10 du Manuel du Spectrum. Les utilisateurs de Spectrum sont particulièrement chanceux car son BASIC inclut une commande permettant de tracer des cercles. Ceci est traité au Chapitre 9 et de ce fait, vous constaterez qu'il n'est pas nécessaire d'entrer dans les détails d'utilisation de SIN, COS et TAN pour tracer des circonférences.

Si vous devez cependant utiliser les fonctions trigonométriques, il est important de vous rappeler que le Spectrum ne mesure pas les angles en degrés, mais en radians. (Le radian en tant qu'unité d'angle est traité dans le Chapitre 9 où il est expliqué avec des dessins d'arcs de cercles sur l'écran.) Pour convertir un angle de degrés en radians utilisez :

```
radians=degres*PI/180
```

et pour convertir des radians en degrés, utilisez :

```
degres=radians*180/PI
```

Les trois fonctions trigonométriques valides sur le Spectrum sont :

SIN - Sinus d'un angle mesuré en radians
COS - Cosinus d'un angle mesuré en radians
TAN - Tangente d'un angle mesuré en radians

Le Spectrum a aussi la capacité de traiter les fonctions inverses de ces trois-là :

ASN - Arc sinus
L'arc sinus d'un nombre est un angle en radians dont le

sinus est égal à ce nombre, x=SIN(ASN(x)). Parce que SIN donne un résultat compris entre +1 et -1, ASN(x) ne peut se calculer que pour des valeurs de "x" comprises entre ces bornes.

ACS - Arc cosinus
L'arc cosinus d'un nombre est un angle en radians dont le cosinus est égal à ce nombre, x=COS(ACS(x)). De même que pour le sinus, le cosinus d'un angle est un nombre compris entre +1 et -1, ce qui impose à "x" dans ACS(x) d'appartenir à cet intervalle.

ATN - Arc tangente
L'arc tangente d'un nombre est un angle en radians dont la tangente est égale à ce nombre, x=TAN(ATN(x)).

Fonctions de chaînes

Nous en avons déjà rencontré quelques-unes au Chapitre 5.

CHR$ - La fonction caractères
CHR$ donnera le "caractère" situé à la énième position dans la liste de caractères du Spectrum. Notez que les mots clés tels que LET comptent pour le Spectrum comme un simple caractère ! La fonction CHR$ retournera tous les caractères que le Spectrum utilise même s'ils ne peuvent pas être imprimés sur l'écran. Ainsi, si vous tapez CHR$(8) ou tout autre nombre supérieur jusqu'à 33 inclus, vous ne verrez rien d'autre qu'un écran blanc.

CODE - Le code d'un caractère
La fonction CODE est l'inverse de la fonction CHR$, car elle retourne la position de tout caractère particulier dans la liste. Par exemple, CODE "A" donne 65 et CHR$(65) donne "A". Si CODE est appliqué à une chaîne de plus d'une lettre c'est le code du premier caractère de la chaîne qui sera donné comme résultat. Si la chaîne est nulle, c'est-à-dire ne contient pas de caractère, le code retourné sera zéro.

LEN - La longueur d'une chaîne
La fonction LEN donne la longueur d'une chaîne. Par exemple, LEN "ordinateur" donne dix et LEN "" (chaîne nulle) donne zéro.

STR$
STR$ est une fonction utile dans des applications évoluées. Elle convertit tout nombre (ou résultat d'expression) en chaîne de caractères imprimables. La fonction STR$ établit ainsi un lien entre les nombres et les chaînes de caractères. Par exemple, "Juillet "+STR$(31) donnera la chaîne "Juillet 31".

VAL - Evalue une expression arithmétique
La fonction VAL est l'inverse de la fonction STR$, elle convertit une chaîne en un nombre. La chaîne peut être n'importe quelle expression arithmétique correcte et le nombre résultant représente la valeur de l'expression. Par exemple, VAL "34" donne 34 et VAL "3+3*6" donne 21.

Fonctions spéciales

Les fonctions RND et INKEY$, donnant respectivement en retour un nombre et une chaîne de caractères, auraient pu être traitées en même temps que les nombres et les chaînes. Cependant, elles sont tellement importantes qu'il est préférable de consacrer un paragraphe entier à chacune de ces deux fontions.

RND
RND est une fonction sans paramètre qui retourne un nombre compris entre 0 et 1, qui peut être considéré comme aléatoire. Savoir qu'un ordinateur peut donner un nombre au hasard sonne toujours un peu comme une contradiction. La confusion vient de l'utilisation du mot hasard. Si vous utilisez un ordinateur pour jouer à un jeu, vous aurez besoin d'une séquence de nombres qui ne soit prévisible par

personne. On peut donc dire des nombres d'une liste qu'ils sont aléatoires s'il n'existe aucune méthode pour en deviner la séquence. Si vous exécutez le programme suivant :

```
10 PRINT RND
20 GOTO 10
```

vous verrez une liste de nombres qui ne suivent aucun modèle évident. (En fait, il y en a un, mais il est si compliqué qu'il faut un Spectrum pour le suivre !) Ce type de hasard est plus justement appelé pseudo-hasard et la fonction RND est un générateur de nombres pseudo-aléatoires. Les nombres qu'elle produit sont répartis uniformément dans la plage de 0 à 1, c'est-à-dire que n'importe quel nombre a la même chance d'apparaître qu'un autre, et de la sorte il n'y a pas de modèle que l'on puisse deviner pour prévoir l'ordre dans lequel les nombres, générés par RND, arriveront.

L'ennui majeur de RND vient du fait qu'il n'est pas fréquent d'avoir besoin de nombres aléatoires compris entre 0 et 1. La meilleure façon d'obtenir un nombre correspondant davantage au besoin est de modifier RND de telle sorte que l'on puisse obtenir un nombre aléatoire entier compris entre 1 et n, où n représente le nombre maximum qui nous intéresse, en utilisant la formule suivante :

```
INT(RND*n)+1
```

Par exemple, si vous voulez programmer les six faces d'un dé, vous choisirez six comme valeur de n et :

```
10 PRINT INT(RND*6)+1
20 GOTO 10
```

imprimera les nombres de 1 à 6 avec approximativement la même fréquence et de telle façon qu'ils ne soient pas évidents à prévoir. Le sujet concernant l'utilisation des nombres aléatoires est trop vaste pour être traité dans ce livre, mais des exemples seront donnés dans les chapitres suivants.

La fonction RND est également particulière parce qu'elle est associée à une autre commande BASIC, RANDOMISE. La liste des nombres que RND produit n'est pas infinie. Elle se répète éventuellement, après 65536 valeurs. Chaque fois que vous branchez le Spectrum il démarre avec la même liste. Si vous voulez vérifier cela, débranchez votre ordinateur et rebranchez-le, puis lancez le programme qui affiche des nombres aléatoires. En répétant cette opération, et en n'oubliant pas de rebrancher la machine à chaque fois, vous obtiendrez toujours les mêmes nombres dans le même ordre. Evidemment ceci n'est pas idéal, car vous pourriez éventuellement apprendre par coeur la séquence produite après la première mise sous tension de l'ordinateur. D'un autre côté pour générer la même séquence, cela oblige à déconnecter la machine, et ce n'est pas là non plus l'idéal. Pour résoudre ces deux problèmes, vous pouvez utiliser la commande RANDOMISE (entrée en appuyant sur la touche notée RAND) pour redémarrer la séquence. La commande RANDOMISE n réinitialise la séquence à partir du énième nombre de la liste du Spectrum. Par exemple si vous entrez :

```
10 RANDOMISE 30
20 PRINT RND
30 GOTO 20
```

vous obtiendrez la même séquence de nombres à chaque fois que vous lancerez ce programme sans avoir à débrancher la machine. Cependant si vous utilisez RANDOMISE 0 ou RANDOMISE sans nombre de départ, le Spectrum utilisera un nombre en rapport avec le temps depuis lequel la machine est connectée pour démarrer la séquence. Pour tester cela, essayez :

```
10 RANDOMISE 0
20 PRINT RND
30 GOTO 10
```

qui imprime le premier nombre dans la séquence produite par RANDOMISE 0. Vous verrez que les nombres imprimés augmentent

lentement, en fonction du temps depuis lequel le Spectrum a été branché. On peut visualiser la plus grande séquence aléatoire que l'on puisse obtenir avec RND et RANDOMISE en exécutant le programme suivant :

```
10 RANDOMISE 0
20 PRINT RND
30 GOTO 20
```

INKEY$

La fonction INKEY$ est implicitement reliée à INPUT par le fait qu'elle peut être utilisée pour lire un simple caractère du clavier. La différence est que INPUT A$ interrompt le programme jusqu'à ce que la touche ENTER ait été actionnée après que l'on ait entré quelque chose à partir du clavier, alors que INKEY$ n'attend rien. Si vous essayez le programme suivant :

```
10 INPUT A$
20 IF A$<>"" THEN PRINT A$
30 GOTO 10
```

vous devrez appuyer sur ENTER avant de voir apparaître des caractères sur l'écran. (Pour arrêter ce programme vous devrez taper STOP.) Cependant si vous modifiez la ligne 10 en :

```
10 LET A$=INKEY$
```

vous verrez apparaître le caractère correspondant à la touche que vous aurez enfoncée. (Pendant le déroulement de la seconde version du programme, essayez d'actionner plusieurs touches en même temps, y compris SHIFT et les autres.) Une autre différence entre INPUT et INKEY$ est que cette dernière n'affiche pas automatiquement quelque chose dans le bas de l'écran.

Chaque fois que le Spectrum rencontre la fonction INKEY$, il examine l'état du clavier. S'il y a déjà une touche

enfoncée, le caractère correspondant est redonné par la fonction, et s'il n'y en a aucune, la fonction redonne la chaîne nulle. Rien d'autre ne peut se produire puisque la fonction INKEY$ n'attend pas qu'une touche soit actionnée.

On se sert principalement de INKEY$ dans les jeux où les touches marquées d'une flèche sont utilisées pour contrôler le déplacement d'un élément sur l'écran. Par exemple :

```
10 LET a$=INKEY$
20 IF a$="" THEN GOTO 10
30 IF a$="5" THEN PRINT "gauche"
40 IF a$="6" THEN PRINT "en bas"
50 IF a$="7" THEN PRINT "en haut"
60 IF a$="8" THEN PRINT "droite"
70 GOTO 10
```

La ligne 10 extrait le caractère correspondant à n'importe quelle touche enfoncée, s'il y en a une. La ligne 20 vérifie si a$ est une chaîne nulle, c'est-à-dire qu'aucune touche n'est enfoncée, et dans ce cas repasse le contrôle à la ligne 10. Ainsi, la boucle formée par les lignes 10 et 20 ne s'arrête que lorsqu'une touche est actionnée. Ensuite les lignes 20 à 50 testent laquelle des touches marquées d'une flèche est enfoncée et imprime le message correspondant. La ligne 70 répète le programme tout entier. Notez que si une touche, autre que celles marquées d'une flèche, est enfoncée, la boucle formée par les lignes 10 et 20 s'arrête mais rien ne s'imprime sur l'écran. Dans le Chapitre 9, un exemple est donné où le même type de programme est utilisé pour guider un point autour de l'écran.

FONCTIONS DEFINIES PAR L'UTILISATEUR - DEF FN ET FN

Dans notre introduction générale aux fonctions, nous avons exposé l'idée qu'une fonction pouvait trouver la valeur maximale parmi deux nombres, mais bien que le Spectrum dispose d'un grand nombre de fonctions, "max" ne figure pas

parmi elles. Le BASIC du Spectrum autorise la définition de nouvelles fonctions mais cela nécessite quelques astuces.

Vous pouvez définir une nouvelle fonction sous forme d'expression. Par exemple, le Spectrum n'ayant pas de fonction d'élévation au carré, on pourra en définir une qui calculera le carré de n'importe quel nombre. Les noms de toutes les fonctions définies par l'utilisateur sont FN suivi d'une lettre, ainsi vous pouvez définir cette nouvelle fonction "FN s" par :

```
DEF FN s(x)=x*x
```

(DEF FN est entré à l'aide d'une seule touche, située dans le coin en haut et à gauche du clavier, la touche 1.) Le mot DEF indique que c'est une définition de fonction. Sa signification est très claire - chaque fois que le Spectrum rencontre la fonction appelée FN, il élève au carré le paramètre entre parenthèses qui la suit. Exemple :

```
LET a=FN s(2)
```

donnerait le résultat 4 stocké dans la variable 'a'. (FN est entré aussi par une seule touche, la 2.) Ceci est correct pour une action en mode immédiat, mais examinons le programme suivant :

```
10 DEF FN s(x)=x*x
20 LET x=3
30 LET z=4
40 LET a=FN s(z)
50 PRINT "x= ";x;" z= ";z;" a= ";a
```

Avant de lancer ce programme, essayez d'évaluer ce que seront les valeurs de 'x', 'z' et 'a'. La réponse exacte est que 'x' sera égal à 3, 'z' à 4, et 'a' à 16. Le fait que 'x' soit utilisé à la fois dans la définition de la fonction FN et dans le programme pourrait entraîner une confusion. En

fait il est à noter que les noms que vous donnez aux paramètres n'ont absolument rien à voir avec les variables utilisées dans le reste du programme. Ainsi, DEF FN s(a)=a*a, DEF FN s(z)=z*z, etc., définissent tous la même fonction ! Dans une définition de fonction, les paramètres montrent seulement la forme que prendra le résultat quand on fera appel à la fonction ainsi définie à ce sujet, ce type de paramètre est appelé paramètre virtuel.

Si vous avez saisi ces concepts, vous devrez être capable de donner le résultat de ce programme :

```
10 DEF FN t(i)=total+1
20 LET total=0
30 PRINT FN t(30)
40 LET total=10
50 PRINT FN t(30)
```

Dans ce cas précis, la variable 'total' de DEF FN n'est pas un paramètre virtuel, mais au contraire une variable du programme principal ('total' de la ligne 10 est le même qu'en ligne 40). La notion de paramètres virtuels deviendra plus facile à comprendre après y avoir réfléchi.

En résumé, les règles à appliquer pour former des fonctions définies par l'utilisateur sont :

1. Chaque fonction que vous utilisez doit être définie à l'aide de la commande DEF FN, n'importe où dans le programme (pas nécessairement avant sa première utilisation).

2. On peut utiliser indifféremment des fonctions de chaînes ou numériques. Les fonctions numériques (celles dont le résultat est un nombre) doivent avoir le nom qui suit FN constitué d'une seule lettre. Les fonctions de chaînes (celles qui ont des caractères comme résultat) doivent avoir leur nom formé d'une lettre suivie du signe $.

3. Une définition de fonction peut admettre plusieurs paramètres virtuels (ou aucun) pour remplacer des numériques

ou des chaînes. Le nom de tous les paramètres virtuels doit être composé d'une lettre suivie du signe $ pour ceux qui remplacent des chaînes. Dans tous les cas, des parenthèses doivent être utilisées, même s'il n'y a aucun paramètre.

4. Toute expression valide du BASIC peut être utilisée dans une définition de fonction.

Ces quatre règles peuvent vous sembler difficiles à retenir, mais elle suivent toujours les mêmes principes. Quelques exemples de fonctions vous aideront à clarifier tout cela :

```
DEF FN m$(a$,i,j)=a$ (i TO i+j-1)
DEF FN l(x)=LN x/LN 10
DEF FN d()=INT(RND*6)+1
```

La première fonction m$ est une fonction de chaîne munie d'un paramètre virtuel chaîne et de deux autres numériques. Ceci extrait "j" caractères à partir de la position "i". Par exemple, m$("bonjour",1,4) donne "bonj". La seconde fonction "l" est une fonction numérique à un seul paramètre virtuel qui calcule le LOG décimal de x (voir les remarques précédentes de la fonction LN). Remarquez que la règle 4 (toute expression valide du BASIC) inclut même les expressions qui font appel aux fonctions. Le troisième et dernier exemple n'a aucun paramètre, mais remarquez que les () sont tout de même nécessaires. Celui-ci donne comme résultat un nombre aléatoire de 1 à 6 (voir les détails de la fonction RND). Si vous faites appel à cette fonction dans une autre instruction, vous devrez toujours utiliser les (), exemple : PRINT d().

Nous pouvons maintenant nous servir de ce que nous avons appris sur les fonctions définies par l'utilisateur, pour créer la fonction "max" qui a été utilisée au début pour introduire le concept de fonctions. En y réfléchissant, il ne semble pas très évident que l'on puisse construire une telle fonction à l'aide de ce que nous venons de voir. Car

après tout, comment peut-on écrire une expression qui puisse choisir l'un des deux nombres ? Si cependant, vous avez bien suivi les explications des expressions conditionnelles du Chapitre 4, et si vous vous rappelez comment elles évaluent à 1 la condition vraie et à 0 le cas contraire, vous ne devriez pas rencontrer de difficulté pour comprendre la solution de ce problème :

```
10 DEF FN m(a,b)=a*(a>=b)+b*(a<b)
20 INPUT a
30 INPUT b
40 PRINT FN m(a,b)
50 GOTO 20
```

Si vous faites tourner ce programme, vous verrez que la ligne 40 imprime toujours le plus grand des deux nombres entrés en ligne 20 et 30. Seule, une des deux conditions (a>=b) et (a<b) peut être vraie, donc l'une donnera 1 et l'autre 0. Comme tout nombre multiplié par zéro donne zéro, ceci explique comment la fonction trouve le résultat. Une nouvelle fois, cet exemple souligne le fait que n'importe quelle expression BASIC peut être utilisée dans une définition de fonction.

SOUS-PROGRAMMES GOSUB ET RETURN

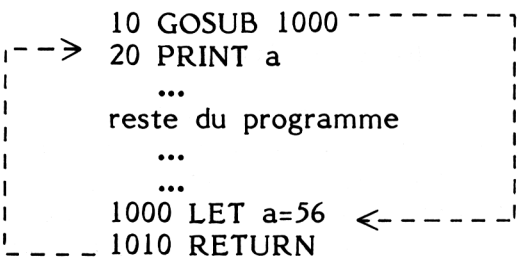
Le fait de pouvoir créer de nouvelles opérations en définissant des fonctions est très puissant, mais il est déjà possible de voir que c'est insuffisant. Quelquefois, il serait commode de pouvoir donner un nom, non seulement à une ligne de BASIC sous forme de fonction définie par l'utilisateur, mais à un groupe donné de lignes. Ceci représente la notion de sous-programme. En effet, un sous-programme n'est rien d'autre qu'un groupe d'instructions BASIC qui peut être utilisé aussi souvent qu'on le veut, en spécifiant simplement son nom. (De la même façon que l'on peut utiliser une simple ligne de BASIC aussi souvent que

souhaité en faisant appel au nom d'une fonction définie par l'utilisateur.) L'ennui avec les sous-programmes BASIC est qu'ils sont limités. Vous ne pouvez même pas donner un nom d'une lettre à un sous-programme. Il doit être référencé par le numéro de sa première ligne et il n'y a aucune possibilité d'assigner des paramètres de quelques sortes qu'ils soient. Mais, même avec ces restrictions, le sous-programme BASIC reste intéressant à connaître et à utiliser.

Si les lignes de BASIC qui constituent le sous-programme démarrent à la ligne n, vous pouvez y faire appel par :

```
GOSUB n
```

dans lequel GOSUB est la contraction de GO to SUBroutine (aller au sous-programme). L'action de GOSUB est très semblable à GOTO, par le fait qu'il transfère le contrôle à la ligne n. La différence entre GOTO et GOSUB est que GOSUB oblige le Spectrum à mémoriser le numéro de ligne du GOSUB dans une zone spéciale de mémoire. Ce numéro mémorisé est utilisé par l'ordre RETURN pour transférer le contrôle à la ligne suivant le GOSUB lorsque le sous-programme est terminé. Par exemple l'effet d'un GOSUB et RETURN sur le flux de contrôle est :



La ligne 10 transfère le contrôle au sous-programme qui démarre à 1000, lequel stocke simplement 56 dans la variable a. L'ordre RETURN en ligne 1010 termine le sous-programme et transfère automatiquement le contrôle à la ligne 20.

Un sous-programme vous permet d'utiliser n'importe quelles instructions BASIC, placées n'importe où dans un programme. En particulier, rien n'interdit à un sous-programme d'en appeler un autre. Dans ce cas, le comportement reste le même, à savoir que le dernier RETURN renverra le contrôle à l'instruction placée immédiatement après le dernier GOSUB.

Cette paire d'instructions, GOSUB et RETURN, constitue la seule chose nécessaire à la définition d'un sous-programme BASIC. Le fait que toutes les variables d'un sous-programme soient les mêmes que dans le reste du programme et qu'il n'y ait aucun paramètre, comme signalé précédemment, peut vous faire croire que les sous-programmes ne sont pas très utiles, mais ce serait une grande erreur !

UTILISATION DES SOUS-PROGRAMMES

Si vous lisez le Chapitre 5 du Manuel du Spectrum, vous serez convaincu que l'utilisation d'un sous-programme est justifiée, à chaque fois que vous aurez besoin de répéter plusieurs fois la même séquence d'instructions dans un programme. Ainsi, si vous avez besoin d'imprimer plusieurs fois le même message dans un grand programme, il sera préférable de remplacer cette ligne par un sous-programme. Bien que cela soit une utilisation fréquente des sous-programmes, il arrive souvent que l'on s'en serve même si une séquence particulière n'est employée qu'une fois dans un programme ! La raison pour laquelle on utilise un sous-programme dans ce cas est que cela rend le programme principal plus clair, donc plus facile à comprendre et à mettre au point. Des exemples d'utilisation de sous-programmes seront présentés dans le reste du livre. Si vous découvrez d'autres méthodes de programmation que vous préférez, personne ne vous critiquera ! Tout ce que nous pouvons dire est que beaucoup de programmeurs sont d'accord sur le fait que les sous-programmes sont très pratiques.

7

LES GRAPHIQUES

L'attrait principal du Spectrum est l'utilisation des graphiques et du son ! Ce chapitre commence par présenter les types de graphiques que l'on peut obtenir avec l'ordre PRINT : les graphiques basse résolution. Le Chapitre 8 traite du son, et le Chapitre 9 traite des graphiques haute résolution. Vous ne devez pas vous imaginer que les graphiques haute résolution sont plus utiles que les basse résolution. Ils ne sont pas plus évolués mais simplement différents, et il est souvent étonnant de voir qu'un programme fonctionne mieux et est plus facile à suivre en utilisant les graphiques basse résolution.

CONTROLER L'INSTRUCTION PRINT

Jusqu'ici nous nous sommes servis de l'ordre PRINT pour afficher des nombres ou des chaînes de caractères sur

l'écran. Bien que cela soit suffisant dans la plupart des programmes, on a souvent besoin de contrôler exactement l'endroit où l'on souhaite afficher quelque chose sur l'écran. Dans le Chapitre 3, PRINT a été défini comme :

PRINT <liste d'affichage>

où <liste d'affichage> est une liste d'éléments, séparés par des ";". Le point-virgule n'est nécessaire que pour montrer où se termine un élément et où commence un autre élément. Par exemple, PRINT somme totale essaiera d'afficher le contenu d'une simple variable appelée 'sommetotale' (souvenez-vous que les espaces sont ignorés dans les noms de variables), alors que PRINT somme;totale affichera le contenu des deux variables 'somme' et 'totale' l'un à côté de l'autre sur la même ligne.

Le Spectrum donne en fait la possibilité d'utiliser trois symboles pour séparer les éléments à afficher, chacun ayant un effet différent. Le point-virgule que nous avons utilisé au Chapitre 3 provoque l'affichage des éléments côte à côte sans laisser d'espace entre eux. L'utilisation de la virgule comme séparation a pour effet de se déplacer à la zone d'affichage suivante avant d'imprimer l'élément suivant. L'écran du Spectrum est divisé en deux zones d'affichage : les seize premières positions et les seize dernières d'une ligne. Ainsi, PRINT "a", "b" affichera "a" en colonne 1 et "b" en colonne 17. Le troisième et dernier séparateur est l'apostrophe que le Spectrum interprète comme une instruction lui disant d'aller à la ligne avant d'afficher l'élément suivant. Ainsi :

```
PRINT "a"'"b"
```

a le même effet que :

```
PRINT "a"
PRINT "b"
```

Vous pouvez placer plus d'un séparateur entre deux éléments à imprimer sans modifier l'effet, PRINT "a",,"b" indiquera au Spectrum de se déplacer de deux zones avant d'afficher "b". Le cas le plus important d'utilisation de séparateurs multiples se rencontre avec l'apostrophe lorsque l'on souhaite laisser plusieurs lignes blanches. Il y a un cas particulier qu'il vaut mieux commenter : si n'importe lequel des trois séparateurs est placé en fin de liste d'affichage, le retour automatique à la ligne suivante est supprimé. Par exemple :

```
PRINT "a",
PRINT "b"
```

revient au même que PRINT "a","b" et

```
PRINT "a";
PRINT "b"
```

est équivalent à PRINT "a";"b"

Bien entendu, une apostrophe à la fin d'un ordre PRINT supprime le retour automatique à la ligne, mais l'apostrophe elle-même force le retour, ainsi n'y-a-t'il pas de différence notable entre PRINT "a" et PRINT "a'".

Pour résumer :

Séparateur	Effet
;	affiche l'élément suivant sans laisser d'espace
,	déplace à la zone d'affichage suivante
'	renvoie à la ligne suivante

FONCTIONS D'AFFICHAGE - TAB ET AT

L'utilisation des séparateurs d'éléments à afficher accroît certainement notre contrôle de l'endroit où ils ils doivent l'être mais nous ne savons pas encore afficher un élément dans une colonne et une ligne données. Pour cela, nous avons besoin d'autre chose que des séparateurs. Deux fonctions spéciales, TAB et AT, sont prévues pour contrôler exactement la place où doit débuter l'affichage d'un élément quelconque. (Elles sont spéciales en ce sens qu'elles sont écrites comme des fonctions ; elles ne produisent aucune valeur et ne peuvent être utilisées qu'en complément d'un ordre PRINT.) TAB est utilisée pour contrôler la position horizontale sur une ligne, et AT est une fonction générale qui permet de pointer n'importe où sur l'écran.

La forme générale de TAB est :

TAB <expression arithmétique>

et son effet est de se placer à la position d'affichage sur la colonne indiquée par la valeur de <expression arithmétique>. Exemple :

```
PRINT "a";TAB 10;"b"
```

affichera "a" en colonne 1 et "b" en colonne 10. Vous pouvez utiliser autant de TAB que vous le désirez dans un ordre PRINT.

```
PRINT "a";TAB 10;"b";TAB 20;"c"
```

affichera "a" en colonne 1, "b" en 10 et "c" en 20.

Deux problèmes peuvent se poser en utilisant TAB. Q'arrive-t'il si vous utilisez une valeur TAB correspondant à une colonne inférieure à la dernière affectée, ou bien si vous précisez une colonne inexistante telle que TAB 35 (il y a seulement 32 colonnes) ? La réponse à la première question est que le Spectrum placera le champ sur la bonne colonne,

mais sur la ligne suivante. Vous pouvez vérifier cela en faisant :

```
10 PRINT "a";TAB 15;"b";TAB 10;"c"
```

qui imprimera "a" en colonne 1, "b" en colonne 15 et "c" en colonne 10 de la ligne suivante. La réponse à la deuxième question est que la machine soustrait 32 à la valeur indiquée de la colonne, jusqu'à ce qu'elle obtienne un nombre compris entre 0 et 32. Par exemple TAB 46 est identique à TAB 14, car 46-32=14, et TAB 126 a le même effet que TAB 30 car 126-32=94, 94-32=62, 62-32=30.

La fonction d'affichage la plus puissante est AT, car elle permet de se positionner à la fois sur une ligne et une colonne. La forme générale est :

```
AT <expression arithmétique 1>,<expression
   arithmétique 2>
```

La valeur de <expression arithmétique 1> représente le numéro de ligne où sera affiché le prochain élément, et <expression arithmétique 2> contient le numéro de colonne. (Si les valeurs de lignes et de colonnes sont en dehors de l'écran, vous provoquerez un message d'erreur out of screen, hors de l'écran.) A la différence de TAB, dont la numérotation de colonnes est 1 à 32, celle-ci varie de 0 à 31 dans la fonction AT. De la même manière, la numérotation de lignes varie de 0 pour la ligne du haut à 21 pour la ligne du bas. Exemple :

```
PRINT AT 3,5;"a"
```

affichera "a" en ligne 3 (c'est-à-dire la quatrième ligne en partant du haut) et colonne 5 (c'est-à-dire la sixième position). (Cela devrait vous sembler moins difficile en utilisant des coordonnées x, y telles que "x" représente la distance horizontale et "y" la distance verticale, puisque vous devez utiliser PRINT AT x,y.)

Une différence intéressante entre TAB et AT est que TAB efface tout le reste de l'écan à partir de la position courante d'affichage. Exemple :

```
10 PRINT "**********************************"
20 PRINT AT 0,0;
30 PRINT TAB(10);"a"
```

Le nombre d'astérisques importe peu. Ils ne sont là que pour montrer l'effet de TAB. Deux points sont intéressants dans cet exemple. Premièrement vous pouvez utiliser AT pour placer la position courante d'affichage n'importe où sur l'écran, sans qu'il soit nécessaire d'afficher quelque chose en même temps. Deuxièmement la ligne 10 montre que TAB (10) efface tous les astérisques depuis le début de la ligne 0 colonne 10. Si vous voulez voir ce que produit AT (il n'efface pas les astérisques mais place "a" en colonne 10), remplacez la ligne 30 par :

```
30 PRINT AT 0,10;"a"
```

Vous pouvez mettre autant de AT que vous souhaitez dans une liste d'affichage, chacun affectant la position d'affichage à l'endroit indiqué avant de traiter l'élément suivant. Non seulement vous pouvez avoir plusieurs AT dans une liste d'affichage, mais vous pouvez mélanger AT, TAB et tous les autres symboles de contrôle d'affichage à votre gré. Par exemple :

```
10 PRINT AT 10,5;"a";TAB(5);"a";AT 3,10;"b""c"
```

est accepté par votre Spectrum sans problème.

A titre d'exercice sur AT, essayez d'écrire un programme qui dessine un carré sur l'écran avec des astérisques. Au cas où vous auriez des problèmes, l'une des solutions possibles (il y a toujours plusieurs manières d'écrire un

programme) est :

```
10 FOR i=0 TO 10
20 PRINT AT 5,i+10;"*"
30 PRINT AT 15;i+10;"*"
40 PRINT AT i+5,10;"*"
50 PRINT AT i+5,20;"*"
60 NEXT i
```

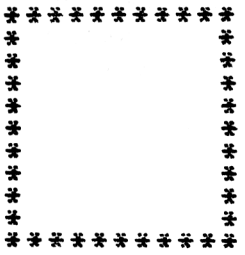


Figure 7.1 : Astérisques formant un carré

Les lignes 20 et 30 impriment les deux lignes horizontales, 40 et 50 les deux colonnes verticales d'astérisques.

ECRAN PLEIN - CLS ET LE DEFILEMENT

L'écran du Spectrum dispose de 22 lignes de 32 caractères, et tôt ou tard vous voudrez afficher d'autres informations alors que toute la place sera occupée. Si vous en avez terminé avec ce qui est déjà sur l'écran, vous pouvez utiliser l'ordre CLS (CLear Screen, effacement de l'écran), qui nettoie simplement l'écran et replace la position courante d'affichage dans le coin en haut et à gauche de l'écran.

Si vous affichez des choses de haut en bas sur l'écran, il arrivera un moment où le bas de l'écran sera atteint. Si vous essayez alors d'afficher une autre ligne, le Spectrum vous demandera Scroll ? y/n (défilement ? oui/non) et

attendra votre réponse. Quelle que soit votre réponse, à part "n", SPACE (espace) ou STOP, l'écran entier remontera d'une ligne en perdant celle du haut mais en dégageant celle du bas sur laquelle l'information suivante sera affichée. Cette technique qui consiste à déplacer l'écran d'une ligne vers le haut est appelée scrolling (défilement), et a été brièvement présentée au Chapitre 2 avec les listes de programmes. Après cette première remontée, de nouvelles lignes sont introduites suivant le même principe, chaque fois qu'un PRINT essaie de dépasser le bas de l'écran. Après 22 remontées, la ligne créée par le premier défilement est prête à disparaître à son tour, et le Spectrum demande encore une fois si l'on veut continuer à défiler ou non. De cette façon, l'ordinateur est certain que vous avez le temps de voir chaque contenu de l'écran, avant que les lignes ne disparaissent par le haut. Pour vérifier cela, essayez par exemple :

```
10 PRINT RND
20 GOTO 10
```

Si pour une quelconque raison vous désirez forcer le défilement avant que l'écran ne soit plein, vous pouvez utiliser :

```
PRINT AT 21,0"
```

qui aura pour effet de commencer à placer la position courante en ligne 21 puis essaiera de la déplacer d'une ligne de plus.

CONTROLER INPUT

Il peut être surprenant de découvrir que l'ordre INPUT peut utiliser tous les éléments de PRINT. En tous cas, c'est une agréable découverte qui permet d'accroître considérablement les formes que nous pouvons donner aux

entrées, sur l'écran. La forme générale de INPUT est :

INPUT <Liste d'affichage>

Il y a cependant quelques règles additionnelles ainsi que d'importantes différences dans la manière dont la liste d'affichage est interprétée avec l'ordre INPUT. La première différence est qu'il est nécessaire d'indiquer quelles sont les variables dont les valeurs doivent être modifiées par INPUT, et lesquelles doivent être affichées comme partie de la question. La règle que le Spectrum utilise est qu'aucun élément d'affichage commençant par une lettre ne doit être traité comme variable d'entrée. (Remarquez que TAB et AT ne sont pas en cause, car ils sont obtenus directement à partir d'une touche.) Par exemple, l'ordre INPUT :

```
10 INPUT "a=";a;"b=";b
```

affichera "a=", attendra que vous donniez une valeur pour a, puis affichera "b=" et attendra jusqu'à ce que vous tapiez une valeur pour b. Vous pouvez contourner la règle de la première lettre en mettant les variables ou expressions entre parenthèses. Par exemple :

```
10 LET a=10
20 INPUT "a=";(a);"b=";b
30 PRINT a,b
```

affichera la valeur de a sur la ligne du bas de l'écran et attendra que vous entriez la valeur de b.

A présent, vous avez remarqué la manière étrange dont PRINT et INPUT opèrent dans des parties d'écran séparées. Cette division de l'écran s'applique à AT. Si vous utilisez AT dans une instruction PRINT, vous devez compter les lignes du sommet de l'écran, à partir de zéro. Cependant, si vous utilisez AT dans une instruction INPUT, vous devez commencer à compter les lignes à partir du bas de l'écran en ligne zéro jusqu'à la première ligne de la zone d'entrée

(Voir Figure 7.2). Essayez, par exemple :

```
10 INPUT AT 0,5;"ligne du haut";AT 1,5;"ligne
   suivante";a
```

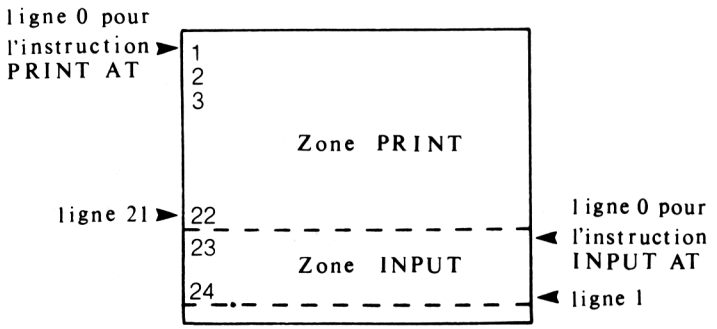


Figure 7.2 : Zones d'écran

Ceci affichera deux propositions, une sur chaque ligne de la zone d'entrée (zone standard INPUT), puis attendra que vous tapiez la valeur de a. A ce stade, vous devriez être agréablement surpris par ce qui arrive en essayant d'imprimer sur des lignes différentes de 0 ou 1 dans la zone d'entrée. En effet le Spectrum fera défiler la zone d'entrée pour amener la ligne sur l'écran. Pour vérifier cela, essayez :

```
10 INPUT AT 10,0;"cette ligne est normalement en dehors
   de l'ecran";AT 0,0;"et ceci est la ligne du haut";a
```

S'il n'y a rien d'imprimé dans la majeure partie de l'écran et qu'un INPUT vienne afficher quelque chose dans cette zone, la zone d'affichage de l'écran (PRINT) remontera pour faire de la place à la zone d'entrée (INPUT). Ceci est facile à comprendre, une fois que vous avez vu ce qui se passe :

```
10 PRINT AT 15,0;"ceci est affiche en ligne 15"
20 PRINT AT 16,0;"ceci est affiche en ligne 16"
30 INPUT AT 20,0;"ceci est affiche en ligne 20
   de la zone d'entree";AT 0,0;"ceci est le
   sommet de la zone d'entree";a
```

Vous pouvez écrire un ordre INPUT qui affiche seulement un message sur l'écran, mais il est improbable que vous le voyiez parce que chaque ordre INPUT efface la zone d'entrée quand il est fini ! Cela veut dire que votre message apparaîtra très peu de temps sur l'écran, puis sera aussitôt effacé, et l'ordinateur traitera l'instruction suivante. Non seulement chaque ordre INPUT nettoie la zone d'entrée de l'écran, mais il la ramène à sa taille normale c'est-à-dire les deux dernières lignes de l'écran.

La division de l'écran en deux parties, la zone PRINT (d'affichage) et la zone INPUT (d'entrée), est considérée comme étant problématique sur le Spectrum (et le ZX81), mais si vous en comprenez le principe cela devient un avantage !

LES CARACTERES GRAPHIQUES

La combinaison de PRINT et AT permet de placer un caractère n'importe où sur l'écran, et comme nous l'avons vu précédemment dans ce chapitre (dans le programme qui dessine un carré), cela peut servir à obtenir des graphiques rudimentaires. Cette possibilité ne trouve vraiment son sens qu'utilisée avec les caractères graphiques du Spectrum. Ceux-ci peuvent être obtenus en appuyant sur Caps Shift et Graphics (touche 9), pour mettre le curseur en G clignotant. Vous pouvez abandonner le mode graphique en appuyant à nouveau sur Graphics. Bien qu'un affichage se produise pour n'importe quelle touche enfoncée, il n'y a que 37 caractères graphiques. Huit de ceux-ci sont écrits en blanc sur la

première rangée de touches, au-dessus des chiffres. Vous pouvez en obtenir huit autres en pressant les mêmes touches en même temps que Caps Shift. Ces derniers sont les mêmes que les premiers mais, en couleur inversée, c'est-à-dire les blancs deviennent noirs et vice versa, mais il est important de comprendre que ce sont des caractères distincts. Ces seize caractères peuvent aussi être obtenus sur l'écran en utilisant leur code dans la fonction CHR$. Pour visualiser ces seize caractères, essayez le programme suivant :

```
10 FOR i=128 TO 136
20 PRINT i,CHR$(i)
30 NEXT i
```

On obtient les 21 autres caractères graphiques par les touches de A à U en étant en mode graphique, et ils correspondent aux codes de 144 à 164. Comme il sera expliqué plus tard, ces caractères peuvent être modifiés afin d'obtenir n'importe quelle forme souhaitée, dans ce cas ils sont définis par l'utilisateur mais ils reproduisent initialement les majuscules. De la sorte en tapant un A graphique, vous verrez un A majuscule s'afficher. A part ces 37 caractères graphiques toutes les autres touches en mode graphique produisent la même chose que dans les autres modes. Pour être précis, n'importe quelle touche actionnée en mode graphique, produit un caractère décalé de 96 positions dans la liste des caractères. En d'autres termes, si une touche permet d'obtenir le caractère CHR$(n), elle produira CHR$(n+96) en mode graphique.

A part le fait d'être entrés dans un mode différent, les caractères graphiques se comportent comme les autres. Ils peuvent être utilisés dans un ordre PRINT ou dans des chaînes. Par exemple, le programme qui décrivait un carré à l'aide d'astérisques peut être modifié avec des caractères graphiques pour obtenir une meilleure qualité d'impression :

```
10 FOR i=1 TO 9
```

```
20 PRINT AT 5,i+10;"[3]"
30 PRINT AT 15,i+10;"[3]"
40 PRINT AT i+5,10;"[5]"
50 PRINT AT i+5,20;"[5]"
60 NEXT i
```

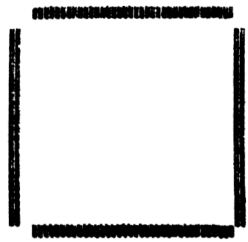


Figure 7.3 : Carré utilisant des caractères graphiques

(Comme nous l'avons expliqué au Chapitre 2, à cause de la difficulté d'impression des caractères graphiques sur un livre, ils sont représentés ici par la lettre se trouvant sur la touche, imprimée entre crochets. Par exemple, [1] désigne le caractère graphique correspondant à la touche 1. Lorsque l'utilisation de Caps Shift est reprise pour obtenir un caractère particulier, nous faisons précéder ce dernier par le symbole "↑". Ainsi [↑1] correspond au caractère graphique obtenu en appuyant sur les touches 1 et Caps Shift.) L'ennui dans le carré est que les coins ne figurent pas, et les afficher relève de l'impression des symboles graphiques en forme de L.

Comme vous devez l'imaginer, il est très difficile de dessiner des formes plus sophistiquées. Fort heureusement, à part les traits horizontaux ou verticaux, on ne se sert des caractères graphiques de la ligne supérieure du clavier, qu'exceptionnellement pour obtenir des formes particulières. Si vous vouliez, par exemple, afficher la silhouette d'un bateau dans un jeu, vous pourriez utiliser :

```
PRINT AT y,x;"[↑2][↑3][↑3]"
```

qui dessinera un bateau en ligne y et colonne x.

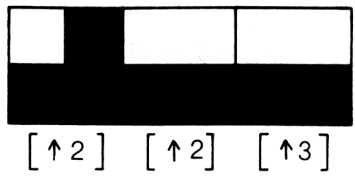


Figure 7.4 : Dessin d'un bateau

Si vous voulez voir un bateau se déplacer sur l'écran, essayez :

```
10 FOR x=0 TO 27
20 PRINT AT 5,x;" [↑2][↑3][↑3]"
30 NEXT x
```

Remarquez l'espace laissé à gauche du premier caractère graphique en ligne 20. Si vous voulez savoir à quoi il sert, essayez de le supprimer !

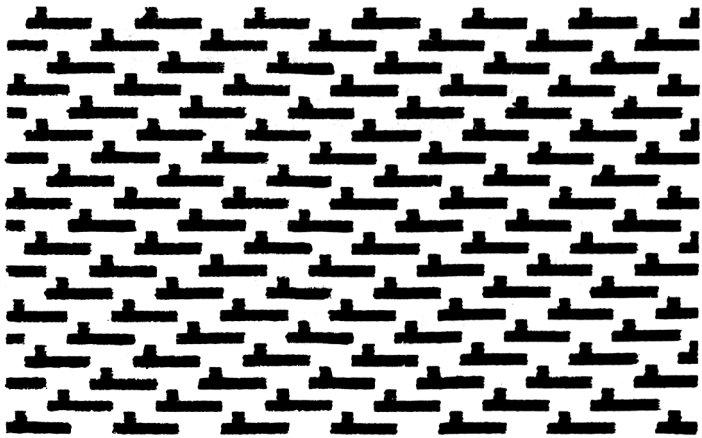


Figure 7.5 : Dessin de bateaux

CARACTERES GRAPHIQUES DEFINIS PAR L'UTILISATEUR

Le nombre de formes qui peuvent être obtenues en combinant les caractères graphiques de la première rangée de touches est très limité. Comment obtenir une forme humaine par exemple ? Heureusement, pour remédier à cela, le Spectrum dispose de 21 caractères modifiables qui permettent de définir toutes les formes que vous désirez.

Avant d'exposer la manière de définir de nouveaux caractères, nous devons regarder comment ils sont produits sur l'écran. Chacun des caractères que le Spectrum affiche est en fait issu d'une grille carrée de 8 points par 8, soit 64 points. Suivant que tels ou tels points de la grille sont affichés en blanc ou en noir, le caractère prend une forme différente. Par analogie avec l'impression ou l'écriture sur papier, les points noirs sont comparables à l'encre, et les blancs au papier. Par exemple, la lettre a est obtenue par le positionnement des points montrés en Figure 7.6 (e pour encre et p pour papier).

```
p p p p p p p p
p p p p p p p p
p p e e e p p p
p p p p p e p p
p p e e e e p p
p e p p p e p p
p p e e e e p p
p p p p p p p p
```

Figure 7.6

Vous ne devez pas trouver évident de voir la lettre a à travers les e et les p, mais cela le devient en remplaçant les e par des astérisques et les p par des blancs comme en Figure 7.7.

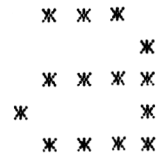


Figure 7.7

Evidemment la forme correspondant à un caractère défini par l'utilisateur doit spécifier de la même manière quels sont les points identifiés à l'encre et ceux qui représentent le papier dans la grille de 8 par 8. Le Spectrum prévoit un moyen très simple pour le faire : les fonctions USR et BIN. Pour remplacer la définition initiale d'un caractère n, n étant une lettre majuscule de A à U, il suffit d'utiliser :

```
USR "n"
```

Prenons l'exemple dans lequel nous voudrions définir un nouveau caractère pour composer une forme humaine, et l'obtenir à partir de la touche [M] (M graphique). Nous utiliserions la fonction :

```
USR "M"
```

La définition du nouveau caractère doit être faite ligne par ligne. Chacune des lignes de la grille peut être écrite sous forme de séquence de huit chiffres, dans laquelle 0 représente le point papier et 1 le point encre. Ainsi, la ligne de points ppeeeppp s'écrirait 00111000. En utilisant cette représentation, chaque ligne du nouveau caractère peut être redéfinie par :

```
POKE USR "M"+1,BIN <sequence de points>
```

dans laquelle "l" est le numéro de ligne à définir (la première ligne étant numérotée 0), et <séquence de points> les 0 et les 1 identifiant les parties encre et papier. L'ordre POKE et la fonction BIN seront expliqués au Chapitre 10, mais savoir exactement ce qu'ils font n'a aucune importance pour définir des caractères ; ils sont toujours exploités de la même manière. Pour définir complètement un caractère, vous devez utiliser au plus huit fois l'instruction décrite précédemment (une fois pour chaque ligne), mais souvenez-vous qu'il n'est pas obligatoire de modifier toutes les lignes d'une définition existante.

Tout ceci sera plus simple après un exemple. On pourrait définir le caractère représentant un petit bonhomme avec la séquence de points suivante :

```
00011000
00011000
11111111
00111100
00111100
00100100
00100100
```

dans laquelle 1 signifie point encre et 0 point papier. Ceci peut être affecté ligne par ligne à la touche [M] graphique de la façon suivante :

```
10 POKE USR "M"  ,BIN 00011000
20 POKE USR "M"+1,BIN 00011000
30 POKE USR "M"+2,BIN 11111111
40 POKE USR "M"+3,BIN 00111100
50 POKE USR "M"+4,BIN 00111100
60 POKE USR "M"+5,BIN 00100100
70 POKE USR "M"+6,BIN 00100100
80 POKE USR "M"+7,BIN 00100100
90 PRINT "[M]"
```

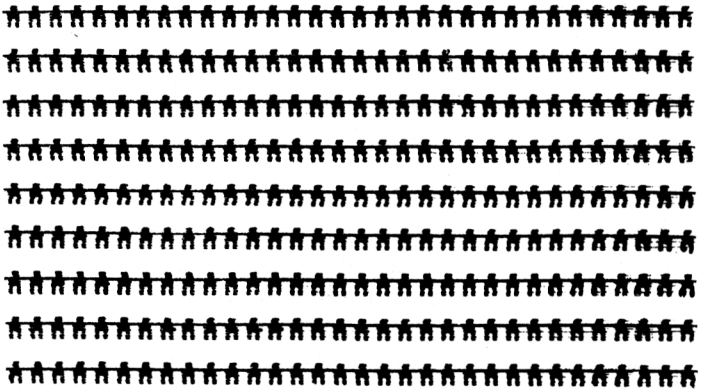


Figure 7.8 : Bonshommes

Remarquez qu'après avoir lancé ce programme, le [M] de la ligne 90 affiche un petit bonhomme. La nouvelle définition de ce caractère est conservée, jusqu'à la prochaine modification ou arrêt de la machine.

Les caractères définis par l'utilisateur sont particulièrement utiles dans les programmes de jeux, pour obtenir des formes spéciales. Toutefois, ils sont aussi utilisés pour montrer sur l'écran des fonctions mathématiques ou des formules de chimie.

MODIFIER L'AFFICHAGE - INVERSE ET OVER

Les deux commandes INVERSE et OVER ne créent ou ne donnent accès à aucun nouveau caractère, mais elles permettent de modifier la façon dont apparaît le caractère sur l'écran. La commande INVERSE est sans doute la plus simple à comprendre.

```
INVERSE 1
```

a pour effet d'échanger les points d'encre et de papier de tout caractère affiché, tel que l'encre devient papier et réciproquement. Ce qui revient à dire qu'au lieu de voir

apparaître les caractères en noir sur fond blanc, vous les verrez en blanc sur fond noir. Pour revenir à l'état initial, il faut réutiliser :

```
INVERSE 0
```

Vous pouvez assimiler le 0 et le 1 qui suivent le mot INVERSE, à un interrupteur marche/arrêt. Exemple :

```
10 INVERSE 1
20 PRINT "ceci est inverse"
30 INVERSE 0
40 PRINT "ceci est le retour
   a l'affichage normal"
```

Bien que INVERSE ne soit pas difficile à comprendre, vous remarquerez deux points importants : INVERSE ne crée aucun caractère nouveau, mais il en modifie l'affichage en changeant l'encre en papier et vice versa. L'effet produit par INVERSE 1 se prolonge jusqu'à ce que la machine rencontre INVERSE 0 ou qu'on l'éteigne. En conséquence, si vous interrompez un programme avant qu'il n'ait rencontré une autre instruction INVERSE qui remette tout dans l'ordre, tous vos listings, etc., apparaîtront en mode inversé. La solution de ce problème consiste à entrer INVERSE 0 en mode immédiat (c'est-à-dire sans numéro de ligne).

La commande OVER est très voisine de INVERSE, en ce sens qu'elle change aussi l'encre en papier mais elle est un peu plus compliquée à comprendre :

```
OVER 1
```

a pour effet de modifier ou non l'affichage des points encre ou papier en fonction de ce qui est sur l'écran. Si le nouveau et l'ancien point sont identiques, le point affiché est du type papier. S'ils sont différents, alors le point est de type encre. Ceci peut paraître compliqué, mais nous pouvons le résumer facilement de la façon suivante :

nouveau point	ancien point	affichage
papier	papier	papier
papier	encre	encre
encre	papier	encre
encre	encre	papier

On peut s'en souvenir en pensant : deux éléments identiques donnent un point papier et deux éléments différents donnent un point encre. (Cette fonction est habituellement connue sous le nom de OU exclusif de l'ancien point avec le nouveau (Voir Chapitre 10). Pour voir l'effet produit, essayez le programme suivant :

```
10 OVER 1
20 PRINT AT 0,0;"souligne"
30 PRINT AT 0,0;"_______"
```

Vous devez voir le message de la ligne 20 souligné. Pour que ceci fonctionne il faut que la dernière ligne de points de chaque caractère représente une ligne de points papier et que la dernière ligne du caractère de soulignement soit composée de points encre. D'après la règle donnée précédemment, papier et encre donnent de l'encre, le caractère de soulignement doit apparaître en dessous de chaque caractère existant sur l'écran. La raison pour laquelle OVER (par-dessus) est ainsi appelé est que l'effet de cette commande rappelle la surimpression sur une machine à écrire, bien que cette assimilation ne puisse être faite en totalité ! Par exemple :

```
10 OVER 1
20 PRINT AT 0,0;"/"
30 PRINT AT 0,0;"/"
```

Ceci produirait la lettre O barrée, sur une machine à écrire, alors que dans notre cas, deux encres donnant du papier, les points communs à la barre / et la lettre O

apparaîtront en points blancs. Bien que cela paraisse problématique, on peut l'utiliser pour obtenir des effets spéciaux. Exemple :

```
10 OVER 1
20 PRINT AT 0,0;"*"
30 GOTO 20
```

La première fois que l'astérisque est affiché, il apparaît sur l'écran car tous les points initiaux sont du type papier. La seconde fois, il disparaît car tous les points affichés à la même place sont du type encre. Ceci est répété chaque fois que l'astérisque est affiché, produisant ainsi l'effet clignotant observé.

Il est important de noter que OVER au même titre que INVERSE ne produit aucun nouveau caractère, mais modifie seulement la manière dont ils sont affichés, et prolonge les effets jusqu'à ce qu'ils soient annulés par :

```
OVER 0
```

ou par l'arrêt de la machine.

ATTRIBUTS DE CARACTERES - FLASH ET BRIGHT

Jusque-là, nous n'avons étudié que la position et la forme des caractères affichés sur l'écran. Cependant, il existe d'autres choses que nous pouvons contrôler dans la manière dont le Spectrum affiche les caractères. Les commandes FLASH (clignotement) et BRIGHT (brillance) peuvent être utilisées pour modifier les caractéristiques d'affichage de n'importe quel caractère.

```
FLASH 1
```

a pour effet de faire clignoter tous les caractères associés à cette commande et

BRIGHT 1

rend les caractères surbrillants par rapport à la normale. Comme pour OVER et INVERSE les effets de FLASH et BRIGHT peuvent être annulés en remplaçant 1 par 0. Ces deux commandes sont plus faciles à illustrer qu'à expliquer, aussi essayez le programme suivant :

```
10 FLASH 1
20 PRINT "caracteres clignotants"
30 FLASH 0
40 BRIGHT 1
50 PRINT "caracteres surbrillants"
60 BRIGHT 0
```

Le message de la ligne 20 clignotera et celui de la ligne 50 sera plus brillant que le pourtour de l'écran. L'effet de FLASH et BRIGHT persiste jusqu'à ce que vous le supprimiez ou que vous mettiez l'ordinateur hors tension.

COULEUR - BORDER, INK ET PAPER

Vous êtes peut-être surpris de constater qu'il nous aura fallu un certain temps avant d'aborder le sujet passionnant qu'est la couleur ! La raison en est qu'il est beaucoup plus facile de comprendre les commandes couleur du Spectrum, après que le concept suivant lequel les caractères sont composés de points encre et papier a été introduit. Le Spectrum peut afficher huit couleurs différentes référencées par les nombres de 0 à 7 :

0 - noir	4 - vert
1 - bleu	5 - bleu ciel
2 - rouge	6 - jaune
3 - magenta	7 - blanc

Il n'est pas nécessaire de retenir ces chiffres, car les noms des couleurs figurent (dans les teintes correspondantes) au-dessus des touches numériques. Remarquez que les chiffres les plus petits correspondent aux couleurs les plus foncées. Si vous n'utilisez pas un récepteur de télévision couleur, celles-ci apparaîtront en dégradé de gris, les plus sombres correspondant aux chiffres les plus bas.

Le moyen le plus simple de voir la palette des couleurs du Spectrum est d'utiliser la commande BORDER. La partie de l'écran sur laquelle le Spectrum ne peut imprimer est appelée la bordure (BORDER) et bien qu'il soit impossible d'y afficher quelque chose, il est possible d'en déterminer la couleur (la bordure est délimitée par le haut et les bords d'écran, et par la ligne supérieure de la zone d'entrée au bas de l'écran). La commande

```
BORDER c
```

aura pour effet de colorer la bordure avec la couleur correspondant au chiffre c. (Notez que d'une manière générale, c peut être une expression arithmétique.) Essayez le programme suivant :

```
10 FOR i=0 TO 7
20 INPUT a$
30 BORDER i
40 NEXT i
```

qui changera la couleur de la bordure à chaque fois que vous presserez la touche ENTER. Pour vérifier que la zone d'entrée fait partie de la bordure, remplacez la ligne 20 par :

```
20 INPUT AT 20,0;a$
```

De même que pour la bordure, il est possible de changer la couleur de tout le reste de l'écran. Cependant, à la

différence de la bordure, ceci nécessite la spécification de deux couleurs, une pour les points encre et l'autre pour les points papier. A la mise sous tension du Spectrum, l'encre est noire et le papier est blanc, mais vous pouvez les modifier de la façon suivante :

```
INK c
```

et

```
PAPER c
```

Ces deux commandes fixent les couleurs des points encre et papier. Essayez le programme suivant :

```
10 INPUT "Couleur de l'encre = ";e
20 INPUT "Couleur du papier = ";p
30 INK e
40 PAPER p
50 PRINT "*";
60 GOTO 10
```

Ce programme vous permet d'imprimer l'astérisque avec n'importe quelles couleurs d'encre et de papier. (Entrez les codes des couleurs en réponse aux questions des lignes 10 et 20.) Remarquez que le Spectrum vous laisse même la possibilité de donner la même couleur pour le papier et l'encre, auquel cas bien sûr vous ne verrez pas l'astérisque ! Il est important de réaliser que l'astérisque est encore là, bien que vous ne puissiez pas le voir. En effet, les points déterminant ce caractère sont présents mais il est impossible de distinguer les points encre des points papier puisqu'ils sont de la même couleur.

Vous savez presque tout sur les commandes couleur du Spectrum, avec lesquelles il est possible de créer d'excellents effets. Toutefois, il est important de connaître les limites de cette méthode simple de contrôle des couleurs. Chaque partie de la grille de huit par huit, formant un caractère, ne peut utiliser que deux couleurs : la couleur de l'encre et celle du papier. Cela signifie que

bien qu'ayant huit couleurs différentes à votre disposition, vous ne pouvez en utiliser plus de deux dans un même caractère. Cela posera un problème lorsque nous regarderons les graphiques haute résolution, au Chapitre 9.

En plus des couleurs de 0 à 7, il existe deux pseudo-couleurs correspondant aux touches 8 et 9. Utiliser le chiffre 8, avec n'importe quelle commande INK, PAPER, BRIGHT ou FLASH, aura pour effet de ne pas modifier l'état des caractères déjà affichés. FLASH 8 laissera clignoter les caractères qui étaient dans cet état, et maintiendra stables ceux qui l'étaient. En quelque sorte, les commandes utilisées avec la couleur 8 produiront un effet de transparence.

La couleur 9 ne peut être utilisée qu'avec les commandes INK et PAPER, et indique simplement au Spectrum d'utiliser une couleur qui contraste par rapport à celle affectée à la position d'impression. Pratiquement, le noir est employé comme contraste avec toutes les couleurs claires (4 à 7), et le blanc pour contraster les couleurs sombres (0 à 3). Pour tester cela, essayez :

```
10 INK 9
20 FOR c=0 TO 7
30 PAPER c
40 PRINT c
50 NEXT c
```

COMMANDES D'AFFICHAGE EN COULEUR

Aucune des commandes d'affichage, OVER, INVERSE, FLASH et BRIGHT, n'est affectée par les couleurs encre et papier. Par exemple, INVERSE échange l'encre en papier sans tenir compte des couleurs correspondantes. BRIGHT donne l'impression de ne modifier que le papier lorsqu'il est utilisé en noir et blanc. Ceci s'explique par le fait que normalement les points papier sont blancs et peuvent donc passer en surbrillance, alors que les points encre étant noirs, il est

impossible de les faire briller ! Par contre, toutes les autres couleurs peuvent être affectées par BRIGHT. Il est possible aussi de combiner les commandes, et généralement dans ce cas le résultat est facile à prévoir. Exemple :

```
10 BRIGHT 1
20 FLASH 1
30 PRINT "brillant clignotant"
```

Vous n'aurez aucune difficulté à comprendre l'affichage couleur du Spectrum, si d'une part vous raisonnez en termes de points encre et papier pour définir les formes sur l'écran, et si vous avez compris d'autre part que les commandes INK et PAPER servent à activer les couleurs. Pour montrer à quel point tout ceci est logique, regardez l'ordre CLS introduit précédemment. CLS efface le contenu de l'écran en le remplissant avec des points papier ; ainsi si vous désirez changer la couleur de l'écran tout entier, essayez :

```
10 INPUT "couleur = ";c
20 PAPER c
30 CLS
40 GOTO 10
```

En exécutant ce programme, vous verrez l'écran changer de couleur instantanément.

COULEURS TEMPORAIRES

Il serait évidemment très intéressant de pouvoir changer de couleurs, seulement le temps d'un ordre PRINT. Ceci peut être réalisé en ajoutant INK ou PAPER comme élément de la liste d'affichage. Voyez :

```
10 PAPER 3
20 INK 6
```

```
30 CLS
40 PRINT "couleur 6"
50 PRINT INK4;"couleur 4"
60 PRINT "couleur 6"
```

Toutes les commandes d'affichage que nous avons rencontrées peuvent être utilisées de la sorte comme faisant partie de la liste d'affichage en limitant ainsi leurs effets aux caractères concernés par l'ordre PRINT. Il en est de même pour l'ordre INPUT. Pour vérifier les effets spéciaux d'affichage, le meilleur moyen est d'utiliser chacune de ces commandes dans un ordre PRINT. De cette manière, vous saurez toujours quelles sont les couleurs que vous obtiendrez sur l'écran.

UTILISATION DES GRAPHIQUES DANS LES JEUX

Nous sommes à présent en mesure d'utiliser des graphiques dans les programmes. Cependant, le programme mettant en oeuvre ce que nous avons appris dans ce chapitre sera mis de côté jusqu'à ce que nous ayons traité le son qui est indispensable pour écrire des programmes de jeux vraiment passionnants.

8
LE SON ET LES JEUX

La commande permettant d'obtenir le son sur le Spectrum est très simple, mais il faut faire preuve d'ingéniosité pour produire des sons intéressants. Dans la première partie de ce chapitre, nous verrons comment obtenir des effets sonores corrects. La seconde partie présente en l'expliquant un exemple de jeu utilisant à la fois les effets sonores et graphiques. Bien que la plus grande partie de ce jeu aurait pu être écrite à la fin du Chapitre 7, vous remarquerez combien il devient plus amusant avec le son.

LES SONS SIMPLES - BEEP

La seule instruction du Spectrum servant à produire des sons est :

BEEP (Bip) <expression arithmétique 1>,<expression arithmétique 2>

dans laquelle <expression arithmétique 1> définit le temps en secondes et <expression arithmétique 2> spécifie la hauteur en demi-tons au-dessous ou au-dessus du do moyen. Ainsi :

```
BEEP 1,0
```

donnera le ton moyen de do pendant exactement une seconde. Notez que le programme s'arrête pendant toute la durée de la note.

Pour écouter toutes les notes possibles, essayez :

```
10 FOR i=69 TO -60 STEP -1
20 BEEP .1,i
30 NEXT i
```

La qualité des notes très aiguës est assez médiocre, elles rappellent davantage un gazouillement plutôt que des notes. Les basses au début produisent un son râpeux, et ensuite ressemblent à des séries de cliquetis. Cela n'est pas réellement surprenant, le cliquetis est le seul bruit que le Spectrum puisse faire ! Les notes sont obtenues grâce à une série de cliquetis très rapide. (Une méthode pour produire des cliquetis est indiquée plus loin dans ce chapitre.)

Si vous connaissez un peu la musique, vous pourrez utiliser BEEP pour écrire vos propres airs. Cependant, si vous voulez écouter tout ce que le Spectrum peut faire par lui-même, essayez :

```
10 BEEP .1,50-INT(RND*100)
20 GOTO 10
```

Le bruit que produit ce programme est intéressant au début mais devient rapidement lassant. L'ennui est que la musique trop aléatoire ne donne pas de sons intéressants. Bien qu'il

soit difficile de structurer suffisamment les sons produits par l'ordinateur pour qu'ils ressemblent à de la musique conventionnelle, vous pouvez améliorer les effets produits comme dans le programme suivant :

```
10 LET n=0
20 LET n=n+SGN(1-2*RND)
30 BEEP .01,n
40 GOTO 20
```

Ce programme joue une longue séquence de notes qui montent ou qui descendent d'un demi-ton au plus et les sons ont juste un peu plus de musicalité que dans le premier programme. En fait, ne croyez pas que ce soit une mauvaise imitation du Vol du bourdon ! Avant d'abandonner la musique aléatoire, il est intéressant d'écouter les effets produits par chacun des programmes ci-dessus, en modifiant la durée des notes. Remplacez les valeurs de la ligne 30 par 1, 0.5, 0.1, 0.01.

Au lieu de musique aléatoire, vous devez penser qu'il serait plus intéressant de pouvoir jouer du Spectrum comme d'un instrument. De fait, il n'est absolument pas compliqué de transformer le clavier du Spectrum en clavier musical ! Exécutez le programme suivant :

```
10 DATA 0,2,4,5,7,9,11,12
20 DIM n(8)
30 FOR i=1 TO 8
40 READ n(i)
50 NEXT i
60 LET a$=INKEY$
70 IF a$="" THEN GOTO 60
80 BEEP .3,n(VAL(a$))
90 GOTO 60
```

Ce programme vous permettra de jouer des notes obtenues à partir des touches numériques 1 à 8, situées sur la première rangée du clavier. Il fonctionne en scrutant continuellement

le clavier à l'aide de INKEY$ (Voir Chapitre 6). Chaque nombre est entré comme variable chaîne et est ensuite converti en nombre par la fonction VAL, puis utilisé dans l'ordre BEEP (ligne 80) pour contrôler la tonalité. Le tableau "n" contient les valeurs de tonalité de chaque note de l'octave au-dessus du do moyen. Vous devriez être capable d'écrire un programme qui fasse en sorte que chaque touche du clavier produise une note différente. Le plus difficile est de trouver un arrangement des touches qui soit pratique pour jouer de la musique !

Au cas où vous voudriez écouter ces sons un peu plus fort, il est intéressant de savoir que les signaux sonores du Spectrum sont aussi utilisables à partir de la sortie cassette. Si vous branchez le magnétophone dont vous vous servez pour sauvegarder et charger vos programmes et que vous enregistrez pendant que le Spectrum joue quelque chose, vous pourrez rembobiner la bande, puis la réécouter en augmentant le son. Si la qualité du son ne vous satisfait pas, rien ne vous empêche de brancher un amplificateur sur la prise MIC.

PROGRAMMATION DES AIRS DE MUSIQUE

Programmer des airs connus ou même des morceaux de votre composition, revient à travailler la séquence de notes et leur durée, ce qui est facile à faire en ayant une partition. Si vous arrivez à trouver le do moyen sur la portée musicale, alors la note de cette ligne correspondra à la tonalité 0. Se déplacer d'une ligne en haut ou en bas sur la portée, revient à augmenter ou à diminuer la tonalité de 2 ou 1 (Voir Figure 8.1). La raison en est que les notes diffèrent, soit d'un ton entier, soit d'un demi-ton seulement. Le modèle des différences entre ton et demi-ton est facile à se rappeler car il correspond exactement à la disposition des touches noires et blanches d'un piano.

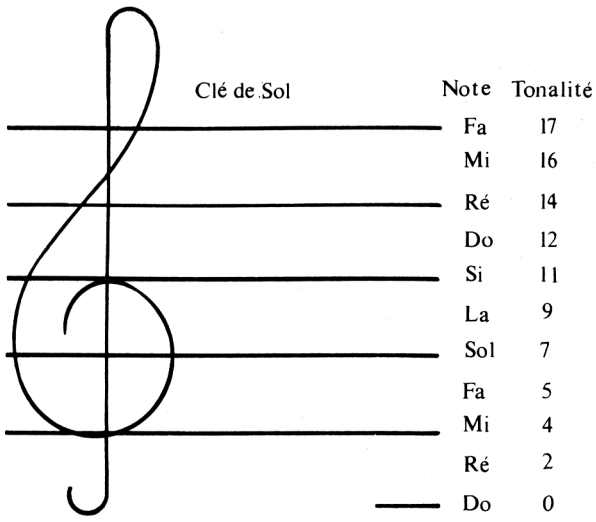


Figure 8.1

A partir du do, cela donne le modèle suivant :

do	ré	mi	fa	sol	la	si	do
ton	ton	1/2	ton	ton	ton	1/2	

Un problème supplémentaire vient du fait que la plupart des musiques utilisent les dièses et les bémols. Ces cas seront facilement résolus dès que vous aurez admis qu'un dièse augmente de 1 la valeur de la note et un bémol la diminue du même nombre. La seule chose dont il faut se souvenir est que, si un signe dièse ou bémol figure au début de la portée (à côté de la clé), toutes les notes correspondantes sur cette portée devront être traitées en dièse ou en bémol.

La Marche funèbre est souvent utilisée pour indiquer à un joueur qu'il a perdu. Les onze premières notes sont montrées en Figure 8.2, et les convertir en valeur de tonalité est assez facile. Le signe dièse (♯) au début s'applique à tous

les fa de la partition. Cette règle est plus évidente en écrivant le nom de chaque note en dessous et ensuite en marquant un signe dièse pour chaque fa. Les valeurs de tonalité sont alors assignées, d'après la Figure 8.1 et en se rappelant qu'il faut ajouter 1 pour un dièse. Lorsque vous convertissez des airs qui ont des bémols dans leur signe de clé vous devez diminuer de 1 la valeur de chaque note concernée. Le résultat des valeurs de tonalité est marqué sous le nom de chaque note en Figure 8.2. A présent, il n'y a plus qu'une seule chose qui vous empêche d'entendre la Marche funèbre, c'est la durée de chaque note. Heureusement, la notation musicale est très logique (après tout c'est l'un des premiers langages programmés !). Le temps est divisé en intervalles, et une note ordinaire telle que la première de la Marche funèbre durerait un intervalle. Le temps que dure une note est écourté proportionnellement au nombre de crochets qui sont dessinés sur sa queue. Chaque crochet divise en deux la longueur de la note, ainsi par exemple, la troisième note ayant deux crochets, le premier l'écourte de la moitié d'un intervalle et le second la réduit à un quart. La dernière complication est qu'une note pointée signifie qu'elle doit être prolongée de la moitié du temps qu'elle devrait durer normalement (maintenant vous allez devenir d'excellents musiciens !). Ainsi, la deuxième note devrait durer un demi-intervalle, mais du fait qu'elle est suivie d'un point, elle doit sonner pendant un demi plus un quart, c'est-à-dire trois quarts. La traduction de cette notation musicale en fractions d'intervalles de temps donne les résultats écrits sous les valeurs de tonalité en Figure 8.2.

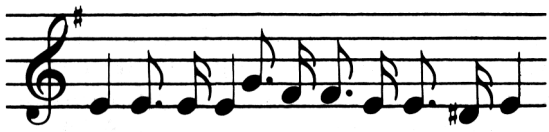
Il y a deux autres notes que l'on ne trouve pas dans la Marche funèbre mais qui sont représentées en Figure 8.3 : la blanche, qui dure 2 fois le temps de base d'une noire, et la ronde, qui dure deux fois le temps d'une blanche soit quatre fois celui d'une noire.

Mi	Mi	Mi	Mi	Sol	Fa♯	Fa♯	Mi	Mi♯	Ré♯	Mi
4	4	4	4	7	6	6	4	4	3	4
1	3/4	1/4	1	3/4	1/4	3/4	1/4	3/4	1/4	1

Figure 8.2 : La Marche funèbre

Note	Durée
𝅝	4
𝅗𝅥	2
♩	1
♪	½
𝅘𝅥𝅯	¼
𝅘𝅥𝅰	⅛

Figure 8.3 : Durée des notes

A présent, nous pouvons commencer la programmation ! Chaque note de cet air est maintenant associée à deux nombres :sa tonalité et sa durée. Ces informations sont plus faciles à mémoriser dans une instruction DATA, pour pouvoir ensuite les lire dans deux variables. (Voir Figure 8.3.)

```
10 DATA 4,1,4,.75,4,.25,4,1,7,.75,6,.25,6,
   .75,4,.25,4,.75,3,.25,4,1,99,99
20 LET temp=1
30 READ p,t
40 IF p=99 THEN STOP
50 BEEP t*temp,p
60 GOTO 30
```

L'instruction DATA se termine par deux fois la valeur 99 pour pouvoir détecter la fin de l'air. La variable 'temps' fixe la longueur du temps fondamental d'un intervalle, 1/4 de seconde dans cet exemple, mais vous pourriez tout aussi bien essayer avec d'autres valeurs.

SILENCE - PAUSE

La seule chose que l'instruction BEEP ne puisse pas faire est d'observer une pause correspondant à un silence. Pour cela, le Spectrum dispose d'une autre instruction qui provoque une interruption de l'exécution pour une période donnée. La forme de cette instruction est :

PAUSE <expression arithmétique>

qui empêchera le Spectrum de travailler pendant le temps donné par la valeur de l'expression arithmétique. Le seul ennui est que ce temps est mesuré en cinquantièmes de seconde, donc :

PAUSE 50

arrêtera le Spectrum pendant une seconde. Ceci n'est pas encore trop problématique car :

```
PAUSE t*50
```

provoquera une pause de 't' secondes.

La commande PAUSE possède une caractéristique particulière : elle se trouve court-circuitée dès que l'on actionne n'importe quelle touche du clavier, ce qui a pour conséquence de faire reprendre l'exécution du programme. En utilisant :

```
PAUSE 0
```

le temps de la pause n'est pas compté et le seul moyen de faire repartir le Spectrum est d'appuyer sur une touche.

Bien que cette commande ait été introduite de manière à obtenir les silences en musique, elle peut être utilisée pour modifier le temps d'exécution d'une partie de programme. En particulier, la commande PAUSE 0 est utilisée pour stopper le Spectrum jusqu'à ce que vous vouliez qu'il continue.

QUELQUES EFFETS SONORES

L'ennui, lorsque l'on utilise le Spectrum pour produire des effets sonores, est qu'au moment même où un bruit provient du haut-parleur, aucune autre fonction ne peut être exécutée. Ainsi, même si vous écrivez un programme qui produit un excellent bruit d'explosion, il sera difficile d'afficher simultanément un graphique représentant l'explosion d'un missile. La meilleure solution dans ce cas est de procéder petit à petit, c'est-à-dire faire un peu de bruit puis déplacer un petit peu le missile, faire un peu plus de bruit, et déplacer un peu plus le missile et ainsi de suite... Un bruit continu accompagnant la trajectoire d'une fusée ne peut pas être obtenu à l'aide du seul BASIC.

Cependant, d'autres bruits peuvent être produits en BASIC sans utiliser l'instruction BEEP. Vous pouvez produire un simple cliquetis par utilisation du sous-programme suivant :

```
1000 LET a=PEEK 23624/8
1010 OUT 254,a-16
1020 OUT 254,a
1030 RETURN
```

Ne vous posez pas trop de questions quant au processus de fonctionnement, car cela fait appel à la conception même du Spectrum, contentez-vous de l'utiliser ! Ce sous-programme produira un cliquetis à chaque fois que les deux instructions OUT 254,a-16 et OUT 254,a seront exécutées.

Essayez par exemple :

```
1000 LET a=PEEK 23624/8
1010 FOR i=1 TO 20
1020 OUT 254,a-16
1030 REM
1040 OUT 254,a
1050 NEXT i
```

qui produira un bruit lent, de tonalité grave. L'instruction REM en ligne 1030 sert simplement à ralentir l'exécution du programme entre deux cliquetis. En augmentant le nombre d'instructions REM, il est possible d'obtenir une sorte de bruit de mitrailleuse.

D'autres effets sonores peuvent être obtenus en combinant des cliquetis avec des notes de courte durée et de fréquences différentes. Les méthodes de programmation à appliquer ne sont pas difficiles, elles nécessitent seulement un peu de patience et de temps pour essayer différentes combinaisons.

ATTAQUE DES SOUCOUPES VOLANTES - SCREEN$, LA FONCTION D'ECRAN

Le jeu listé ci-dessous utilise la plupart des commandes de BASIC qui ont été décrites dans ce chapitre et le Chapitre 7.

```
10 LET t=15
20 LET f=0
30 LET fx=10
40 LET fy=10
50 GOSUB 1000
60 FOR x=0 TO 30
70 GOSUB 2000
80 GOSUB 3000
90 PRINT AT 20,x;"[A]"
100 NEXT x
110 FOR x=30 TO 0 STEP -1
120 GOSUB 2000
130 GOSUB 3000
140 PRINT AT 20 ,x;"[A]"
150 NEXT x
160 GOTO 60

1000 POKE USR "[A]"  ,  BIN
     00000000
1010 POKE USR "[A]"+1, BIN
     00111100
1020 POKE USR "[A]"+2, BIN
     00111100
1030 POKE USR "[A]"+3, BIN
     01111110
1040 INK 2
1050 PAPER 5
1060 CLS
1070 RETURN

2000 LET f$=INKEY$
```

```
2010 IF f$="" THEN RETURN
2020 BEEP .01,10
2030 PRINT AT fy,fx;" "
2040 LET fx=x
2050 LET fy=21
2060 LET f=1
2070 RETURN

3000 LET t=t+RND*2-1
3010 IF t<5 THEN LET t=t+1
3020 IF t>25 THEN LET t=t-1
3030 PRINT AT 5,t;" *** "
3040 IF f=0 THEN PAUSE 5:RETURN
3050 PRINT AT fy,fx;" "
3060 LET fy=fy-1
3070 PRINT AT fy,fx;" "
3080 BEEP .001,50-fy*2
3090 IF fy<4 THEN LET f=0:PRINT AT fy,fx;"":RETURN
3100 IF SCREEN$(fy-1,fx)<>"*" THEN RETURN
3110 GOSUB 4000
3120 RETURN

4000 PRINT AT 5,t+1;FLASH 1;" *** "
4010 BEEP .01,40
4020 LET a=PEEK 23624/8
4030 FOR i=1 TO 20
4040 OUT 254,a-16
4050 OUT 254,a
4060 NEXT i
4070 RETURN
```

La partie est relativement facile à jouer. Une soucoupe volante représentée par trois astérisques se déplace à travers l'écran. Un vaisseau se déplace rapidement d'avant en arrière, en bas de l'écran. Un missile du vaisseau

attaquant peut être tiré à tout moment, en actionnant n'importe quelle touche. Une fois tiré, le missile se déplace vers le haut de l'écran accompagné d'un sifflement, augmentant de tonalité si l'objectif est manqué ou provoquant un bruit d'explosion s'il est atteint. Si une autre touche est actionnée pendant le vol du missile, celui-ci est effacé de l'écran et un nouveau missile est tiré sur la soucoupe.

Le programme a été écrit sous la forme d'un ensemble de sous-programmes et de ce fait peut être compris facilement. Il est plus commode de suivre le corps principal du programme après avoir décrit chaque sous-programme. Le sous-programme de 1000 à 1070 met en place les caractères graphiques définis par l'utilisateur pour le vaisseau attaquant (lignes 1000 à 1030) et fixe également les couleurs de l'encre et du papier (lignes 1040 à 1070). Le sous-programme 2000 à 2070 teste si une touche quelconque a été enfoncée, et si oui, tire le missile. Dans le cas contraire, le contrôle est transféré au corps principal du programme (ligne 2010). Si une touche est actionnée, tout missile déjà existant est effacé de l'écran par une impression en blanc (ligne 2030) à sa position courante qui est mémorisée en fy et fx. Ensuite, la position du missile est initialisée à la position courante du vaisseau attaquant (lignes 2040-2050) et la variable f est initialisée à 1 (ligne 2060) pour indiquer qu'un missile a été tiré et qu'il est en vol. Le sous-programme 3000-3120 déplace la soucoupe au hasard en montant à droite ou à gauche, affiche le missile s'il y en a un en vol et contrôle s'il a touché la soucoupe. Ce sont les lignes 3000 à 3030 qui ont pour fonction de déplacer la soucoupe. Remarquez les contrôles, en lignes 3010 et 3020, pour l'empêcher de déborder l'écran lors de son déplacement. L'impression de la soucoupe en ligne 3030 sert aussi à effacer sa trace précédente grâce aux caractères blancs situés devant et derrière les astérisques (" *** "). Les lignes 3040 à 3080 contrôlent le déplacement du missile. La ligne 3040 teste s'il y a un missile en vol (c'est-à-dire f=1). S'il n'y en a pas, le

contrôle est retourné à la partie principale du programme. La commande PAUSE sert à conserver la même vitesse de déplacement du vaisseau attaquant quand aucun missile n'est en vol. Les lignes 3050 à 3080 déplacent le missile d'une ligne vers le haut de l'écran. La ligne 3050 efface la position précédente du missile et la ligne 3070 affiche sa nouvelle position. La ligne 3080 produit un son dont l'intensité croît en fonction du déplacement du missile. Les lignes 3090 à 3110 testent si le missile a atteint ou non la soucoupe. La ligne 3090 contrôle si le missile a dépassé la soucoupe ; si c'est le cas, un blanc est imprimé pour effacer le missile, et la variable f est remise à zéro indiquant ainsi qu'il n'y a plus de missile en vol. La ligne 3100 utilise la fonction SCREEN$ pour connaître le prochain caractère que le missile va rencontrer. La fonction SCREEN$ n'a pas été traitée jusqu'ici, mais il est très facile de comprendre son fonctionnement. SCREEN$(y,x) donne le caractère affiché sur l'écran en ligne y colonne x. Elle sert en ligne 3100 à vérifier si le caractère situé au-dessus du missile est un astérisque. Si c'est le cas, le missile est prêt d'atteindre la soucoupe et le sous-programme d'explosion est appelé.

Les lignes 4000 à 4070 forment le sous-programme d'explosion. La ligne 4000 fait clignoter la soucoupe pour figurer l'explosion sur l'écran. Les lignes (4020-4060) produisent le bruit sourd décrit dans un paragraphe précédent pour simuler une explosion.

A présent que tous les sous-programmes ont été décrits, il est aisé de comprendre le fonctionnement du corps principal du programme. Les lignes 10 à 40 initialisent les principales variables : t identifie la position horizontale de la soucoupe, f sert à indiquer si un missile a été tiré et fx et fy représentent les coordonnées de la position du missile. Le sous-programme 1000 est appelé ensuite pour initialiser les caractères graphiques définis par l'utilisateur. La fonction principale du programme est réalisée par les deux boucles FOR 60 à 100 et 110 à 150. La première déplace, d'une position vers la droite, le vaisseau

attaquant. Chaque fois que ce dernier se déplace, le sous-programme 2000 est appelé pour "la mise à feu" d'un missile, et le sous-programme 3000 pour déplacer le missile et la soucoupe. La deuxième boucle est identique à la première, à ceci près qu'elle déplace le vaisseau attaquant vers la gauche.

Ceci conclut la description de ce petit programme de jeu. Si vous souhaitez l'étudier davantage, la meilleure façon de vérifier que vous l'avez bien compris est d'essayer de le modifier ! Ce jeu pourrait être amélioré en y ajoutant quelques caractéristiques simples. Vous pourriez, par exemple, ajouter un programme pour mémoriser le nombre de soucoupes atteintes, ou bien limiter le nombre de missiles du vaisseau attaquant, etc. Après tout, la meilleure façon d'apprendre à programmer est encore de programmer !

9

GRAPHIQUES HAUTE RESOLUTION

Comme les Chapitres 7 et 8 ont dû vous en convaincre, le graphisme haute résolution n'est absolument pas indispensable à la plupart des applications. Dessiner des choses, en graphisme haute résolution, vous prendra généralement beaucoup de temps, et de plus vous serez très souvent limité à deux couleurs. Ceci étant dit, vous n'aurez aucune difficulté à comprendre les commandes qui permettent de l'utiliser si vous maîtrisez bien les principes du graphisme basse résolution.

L'ECRAN HAUTE RESOLUTION

Le Spectrum emploie la même méthode pour traiter les graphismes haute et basse résolution. Ceci veut dire qu'il est possible de mélanger le texte et le graphisme haute

résolution, ce qui est un avantage que tous les micro-ordinateurs n'ont pas.

La manière dont fonctionne la haute résolution est facile à comprendre, en considérant les carrés de huit points par huit. Les instructions graphiques basse résolution ne peuvent modifier que globalement un bloc de huit points par huit points. Par contre, les instructions graphiques haute résolution peuvent agir sur un point isolé de n'importe quel bloc. Notons que toutes les règles concernant l'affichage restent valables. La couleur du point, par exemple, est déterminée par les commandes INK et PAPER concernant le bloc auquel il appartient. Les commandes haute résolution permettent d'afficher un point isolé, mais ne permettent pas d'agir sur la couleur ou les autres attributs. Pour afficher un point isolé, il doit exister un moyen de le localiser. A présent, vous devriez théoriquement être capable de faire référence à un point isolé tel que le troisième point d'un caractère particulier en vous servant de la numérotation des positions sur l'écran. Mais il est beaucoup plus facile d'utiliser un nouveau système de référence plus adapté. Comme chaque ligne comporte 32 caractères de 8 points de large, cela fait un total de 32*8, 256 points par lignes. La hauteur comprenant 22 lignes de 8 points, cela donne 22*8, 176 points verticalement. De la sorte, l'écran haute résolution est composé de 256 points horizontaux et 176 verticaux, et on peut isoler n'importe quel point en définissant sa ligne et sa colonne. Les colonnes sont numérotées à partir de zéro qui représente la colonne extrême gauche, et le numéro de colonne appelé coordonnée x varie de 0 à 255. Les lignes sont aussi numérotées à partir de zéro qui identifie la ligne du bas, et le numéro de ligne appelé coordonnée y varie de 0 à 175. (Remarquez que le sens des numéros de ligne est l'inverse de celui des lignes de programme qui partent du haut de l'écran.) Donc, n'importe quel point de l'écran peut être identifié à l'aide de ces deux coordonnées, x et y. On note habituellement ces deux nombres entre parenthèses, le premier étant x. Ainsi (0,0) identifie le coin inférieur gauche, et (255,175) le coin

supérieur droit. Avec un peu de pratique, il devient naturel d'utiliser les coordonnées x et y.

LES COMMANDES GRAPHIQUES - PLOT, DRAW et CIRCLE

Les commandes graphiques sont plus faciles à comprendre quand elles ne fonctionnent qu'avec deux couleurs, c'est donc par là que nous commencerons ! On supposera, dans tout ce qui suit, que les couleurs d'encre (noir) et de papier (blanc) n'ont pas été modifiées ou que l'on a choisi des couleurs suffisamment contrastées à l'aide des instructions INK et PAPER.

La commande haute résolution la plus simple est :

PLOT <expression arithmétique 1>,<expression arithmétique 2>

Cette instruction change le point de coordonnées x=<expression arithmétique 1> et y=<expression arithmétique 2> en un point encre. Essayez le programme suivant pour découvrir comment fonctionnent PLOT et les coordonnées x, y :

```
10 INPUT "x=";x;" y=";y
20 PLOT x,y
30 GOTO 10
```

Si vous donnez à x et y des valeurs qui définissent un point en dehors de l'écran, vous obtiendrez le message d'erreur integer out of range (nombre hors limites). Pour une démonstration automatique, essayez :

```
10 LET x=INT(RND*256)
20 LET y=INT(RND*176)
30 PLOT x,y
40 GOTO 10
```

La commande la plus utile en haute résolution est :

DRAW <expression arithmétique 1>,<expression arithmétique 2>

qui tire un trait. La position de départ du trait est celle du dernier PLOT ou DRAW, et sa position finale est <expression arithmétique 1> vers la droite et <expression arithmétique 2> vers le haut. Si le dernier PLOT était PLOT 0,0 par exemple, la commande :

DRAW 100,100

tirerait un trait de (0,0) à (100,100). Mais si le dernier PLOT avait été PLOT 50,50, le trait aurait débuté en (50,50) et se serait terminé en (150,150). Il est important de remarquer que DRAW utilise des coordonnées complètement différentes de PLOT. Lorsque vous utilisez les commandes haute résolution, imaginez qu'il y a un curseur graphique qui se déplace sur l'écran en fonction de la position courante, de la même façon que le curseur de contrôle qui se déplace avec la position d'impression. Ainsi, la commande PLOT x, y, déplace le curseur graphique au point (x,y). La commande DRAW x,y déplace le curseur graphique de x unités horizontalement et de y unités verticalement, en tirant un trait de la position courante initiale du curseur à la nouvelle. Des commandes telles que RUN, CLEAR, CLS et NEW réinitialisent la position du curseur au point (0,0). La différence la plus flagrante entre DRAW et PLOT apparaît dans la forme suivante :

DRAW -10,10

qui ramène le curseur de 10 positions vers la gauche et 10 en hauteur, alors que les coordonnées négatives sont interdites avec la commande PLOT !

La commande DRAW est très pratique, mais il se produit parfois des cas où il est difficile de tirer un trait entre

deux points. Essayez la combinaison suivante qui trace un trait entre le point (x1,y1) et (x2,y2) :

```
PLOT x1,y1:DRAW x2-x1,y2-y1
```

Pour voir ce que DRAW peut faire, essayez le programme qui suit :

```
10 FOR i=1 TO 175 STEP 4
20 PLOT 0,i
30 DRAW 255-i,-i
40 PLOT i,0
50 DRAW 255-i,i
60 PLOT 0,i-175
70 DRAW 255-i,i
80 PLOT i,175
90 DRAW 255-i,-i
100 NEXT i
```

L'image que donne ce programme est montrée en Figure 9.1.

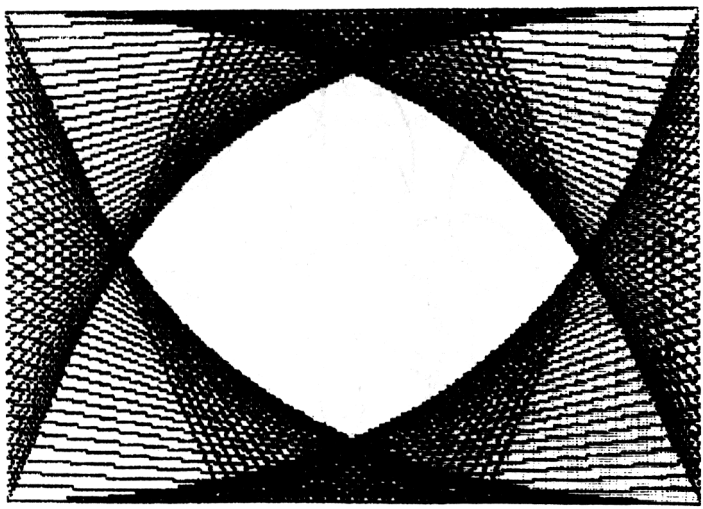


Figure 9.1 : Programme de faisceaux

Vous pouvez constater les différents effets produits, en modifiant la valeur de STEP en ligne 10. Il sera plus facile de comprendre les caractéristiques des extensions de la commande DRAW, après avoir vu la dernière commande haute résolution qui est :

CIRCLE <expression arithmétique 1>,<expression arithmétique 2>,<expression arithmétique 3>

dont le résultat est un cercle centré sur le point (x,y), dans lequel x et y sont les valeurs des deux premières expressions arithmétiques, et de rayon déterminé par <expression arithmétique 3>. Exemple :

CIRCLE 100,50,40

dessine un cercle centré au point (100,50) et de rayon 40. A titre d'exemple de cette commande, le programme suivant dessine des cercles aléatoires, tels qu'illustrés en Figure 9.2 :

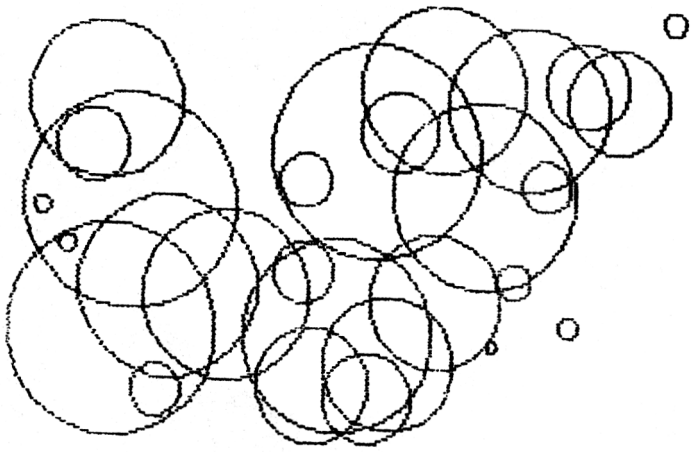


Figure 9.2 : Cercles aléatoires

```
10 LET r=RND*50
20 LET x=r+RND*(255-2*r)
30 LET y=r+RND*(175-2*r)
40 CIRCLE x,y,r
50 GOTO 10
```

Remarquez qu'en dessinant des cercles aléatoires, il faut faire attention à ce qu'ils ne débordent pas de l'écran. Cette précaution est prise dans l'exemple par -2*r en lignes 20 et 30.

Maintenant que nous avons vu la commande CIRCLE, nous pouvons aborder la caractéristique supplémentaire de DRAW, dont nous avons fait état précédemment. En plus des traits, DRAW inclut la possibilité de tracer des arcs de cercle entre deux points. La forme générale de DRAW est :

DRAW <expression arithmétique 1>,<expression arithmétique 2>,<expression arithmétique 3>

La présence de la troisième expression arithmétique ne change rien à la signification des deux premières. Cependant, le troisième paramètre spécifie l'angle utilisé pour déterminer l'arc de cercle à dessiner. Pour comprendre pourquoi un angle est nécessaire pour préciser l'arc à dessiner, imaginez que vous soyez placé au centre d'un cercle. Si vous écartez les bras d'un angle donné, vous déterminez une portion de cercle qui correspond à l'arc que la commande DRAW doit dessiner. Par exemple, 180 degrés spécifient un demi-cercle et 90 degrés un quart de cercle. La seule complication est que le Spectrum mesure les angles en radians plutôt qu'en degrés. Pour convertir des degrés en radians, vous devez multiplier par PI (π), puis diviser par 180. Il résulte de ceci, qu'un angle de PI radians spécifie un demi-cercle et qu'un angle de PI/2 radians donne un quart de cercle, etc. A titre d'exemple, essayez :

```
10 PLOT 100,100:DRAW 50,50,PI/2
```

La commande PLOT place le curseur graphique en (100,100) et DRAW dessine un quart de cercle débutant en (100,100) et se terminant en (150,150). Remarquez que l'arc de cercle n'est déterminé que par le troisième paramètre. La dimension et l'orientation de l'arc de cercle sont déterminées par la position courante du curseur et des deux premiers paramètres.

Vous devez savoir qu'à la différence d'une droite, deux points quelconques déterminent deux arcs de cercle. Si vous imaginez deux points quelconques sur une droite horizontale, vous pouvez alors dessiner un demi-cercle en partant du point gauche vers le point droit en tournant soit dans le sens des aiguilles d'une montre, soit à l'inverse. Le premier cas déterminant le demi-cercle supérieur et le second le demi-cercle inférieur. Cette description correspond exactement à la façon dont le Spectrum résout le problème de savoir quel cercle dessiner. Les angles négatifs déterminent des arcs de cercle dessinés dans le sens des aiguilles d'une montre, et les angles positifs dans le sens inverse.

Comme exemple d'une commande DRAW complète, essayez le programme suivant qui donne un dessin ressemblant à des ficelles emmêlées (Voir Figure 9.3).

Figure 9.3 : Programme de ficelles emmêlées

```
10 PLOT 100,100
20 LET x=25-RND*50
30 LET y=25-RND*50
40 LET p=PI*RND
50 DRAW x,y,p
60 GOTO 20
```

La ligne 10 positionne le curseur graphique au milieu de l'écran, et les lignes 20 à 30 initialisent à des valeurs aléatoires les deux premiers paramètres de DRAW. La ligne 40 détermine aléatoirement l'angle, de telle manière que ce soit une fraction de PI qui indique à la commande DRAW de dessiner des fractions de demi-cercles. En laissant tourner suffisamment longtemps ce programme, il se terminera par un message d'erreur quand les arcs déborderont de l'écran. En général, il est difficile de détecter le moment où un arc de cercle dessiné par une commande DRAW sortira de l'écran.

COULEUR HAUTE RESOLUTION

Toutes les commandes de couleur et d'attribut telles que INK, PAPER et OVER, introduites au Chapitre 7, peuvent être utilisées pour contrôler l'affichage des points haute résolution. Si vous souhaitez obtenir des points bleus sur fond jaune, par exemple, vous n'avez qu'à utiliser INK 1:PAPER 6 avant n'importe quelle commande PLOT, CIRCLE, ou DRAW. De même que pour la commande PRINT, il est possible de spécifier des commandes couleur dans une instruction haute résolution. Dans ce cas, les effets produits par ces commandes couleur n'affecteront que les points spécifiés de l'écran, et de plus seront temporaires. La seule chose à se rappeler est que, à part OVER et INVERSE, les autres commandes couleur ou d'attribut s'appliquent à des caractères entiers. Si vous pointez un point particulier, tel que PLOT INK 4;100,100, tous les points encre du caractère englobant la position 100,100 seront colorés en vert (4). L'illustration la plus importante sur la façon

dont INK et PAPER affectent un bloc entier de caractères consiste à tirer un trait d'une couleur différente du reste de l'écran, à l'aide des commandes DRAW et PAPER. Exemple :

```
10 PAPER 6
20 CLS
30 DRAW PAPER 1;INK 4;100,100
```

L'affichage obtenu sera très irrégulier, car chacun des points spécifiés par la commande a pour effet de changer en bleu (1) la couleur de tous les points papier du caractère ainsi défini. S'il y avait eu des points encre d'une autre couleur, ils auraient été transformés en vert.

Le fait que les couleurs (y compris BRIGHT et FLASH) s'appliquent aux caractères entiers restreint de façon considérable leur utilisation avec les graphiques haute résolution. Si vous tirez deux traits de couleurs différentes, il n'y aura aucun problème, sauf s'ils se rencontrent en un point. Quand cela se produit, la partie du premier trait se trouvant dans la même zone caractère que le point de croisement prend la couleur du second trait. La règle générale est que vous ne pouvez utiliser plus de deux couleurs dans un caractère à une position donnée : la couleur de l'encre et la couleur du papier. Tant que des lignes et des cercles haute résolution restent dans leurs propres zones d'affichage, ils peuvent être colorés dans n'importe laquelle des huit couleurs, mais si certaines de ces figures partagent la même zone, elles doivent avoir alors les mêmes couleurs d'encre et de papier. Pour bon nombre d'applications, cette restriction n'a aucune importance, car elles utilisent des figures dans des zones différentes. Par contre, si ce n'est pas le cas, il n'y a aucun moyen de contourner cette difficulté sur le Spectrum. Pour éviter de tels problèmes, le meilleur moyen est de n'utiliser que deux couleurs en haute résolution.

INVERSE DE PLOT - OVER ET INVERSE

Bien que les commandes couleur INK, PAPER, FLASH et BRIGHT affectent tous les points d'un caractère, OVER et INVERSE permettent d'affecter n'importe quel point particulier de l'écran. Cela signifie que ces dernières sont de vraies commandes haute résolution qui peuvent donc être utilisées dans PLOT, DRAW ou CIRCLE sans que cela ait de conséquences sur les figures déjà affichées sur l'écran. Par exemple INVERSE 1 changera les couleurs d'un point d'encre en papier. Ainsi le programme :

```
10 CIRCLE INVERSE 0;100,100,50
20 CIRCLE INVERSE 1;100,100,50
30 GOTO 10
```

dessine un cercle de points encre, et l'efface en le redessinant en points papier.

La commande OVER est plus difficile à comprendre parce que l'effet produit dépend de ce qui est déjà défini en un point donné. En vous rappelant les règles de fonctionnement de la commande OVER, expliquées au Chapitre 7, vous devriez être à même de voir qu'en haute résolution, OVER 1 fera apparaître un point encre, si elle remplace un point papier, et un point papier dans le cas contraire. C'est très utile en pratique car cela permet d'afficher ou d'effacer des points, sans pour autant altérer les autres figures de l'écran. Pour en voir le fonctionnement, essayez :

```
10 CIRCLE 100,100,50
20 PLOT 0,100
30 DRAW OVER 1;255,0
40 PAUSE 20
50 GOTO 20
```

La première fois, la commande DRAW trace une ligne de points encre traversant le cercle généré par la ligne 10, excepté les points communs au cercle et à la droite, qui, en accord

avec les règles données plus haut, sont des points papier. Après la pause contrôlée par la ligne 40, pour laisser le temps de voir le résultat, les commandes PLOT et DRAW sont à nouveau exécutées. Cette fois, la ligne disparaît car les points encre sont changés en points papier sauf ceux qui coupent le cercle redevenant alors des points encre ! Ceci fait donc apparaître et disparaître alternativement la ligne jusqu'à ce que vous interrompiez le programme. Ainsi, pour obtenir momentanément des lignes ou des cercles, utilisez la commande OVER 1 une première fois pour les afficher, puis une seconde fois pour les effacer.

Il est plus clair de résumer l'emploi en haute résolution des commandes OVER et INVERSE, sur un exemple d'instruction PLOT :

PLOT	donne un point encre.
PLOT INVERSE 1	donne un point encre.
PLOT OVER 1	donne la couleur opposée d'un point déjà affiché, (encre en papier et papier en encre).
PLOT INVERSE 1;OVER 1	n'a aucun autre effet que de placer le curseur graphique.

DECOUVRIR CE QUI EST SUR L'ECRAN - POINT

De la même façon que SCREEN$ pouvait être utilisée pour découvrir quel caractère était affiché en une position donnée, la fonction POINT permet de trouver le type de point utilisé en n'importe quelle position d'un écran haute résolution. La forme générale de la fonction POINT est :

POINT (<expression arithmétique 1>,<expression arithmétique 2>)

dans laquelle <expression arithmétique> 1 et 2 représentent les coordonnées x et y du point considéré. Notez que, comme

dans la plupart des fonctions du Spectrum, les parenthèses sont nécessaires. La valeur retournée par cette fonction est 1 si le point spécifié par ses coordonnées est un point encre et 0 si c'est un point papier. On utilise habituellement POINT en haute résolution dans les mêmes cas où SCREEN$ aurait été employé, comme dans le programme des soucoupes volantes proposé au Chapitre 8.

UTILISATION DES GRAPHIQUES HAUTE RESOLUTION

Bien qu'il n'y ait que quelques commandes graphiques haute résolution, il est difficile de montrer tous les effets que l'on peut obtenir. Comme toujours, le meilleur moyen d'apprendre à s'en servir est de regarder les exemples et d'essayer d'écrire vos propres programmes.

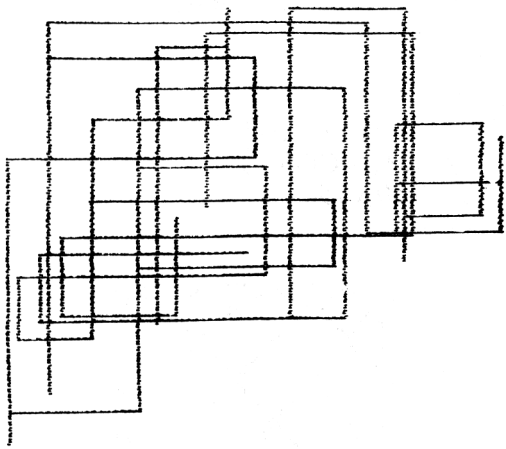


Figure 9.4 : Programme de dessin

Le premier exemple utilise les fonctions INKEY$ pour contrôler la position d'un point sur l'écran. En appuyant sur la touche flèche appropriée, vous pouvez dessiner des formes sur l'écran :

```
10 LET x=127
20 LET y=80
30 LET a$=INKEY$
40 IF a$="5" THEN LET x=x-1
50 IF a$="8" THEN LET x=x+1
60 IF a$="6" THEN LET y=y-1
70 IF a$="7" THEN LET y=y+1
80 PLOT x,y
80 GOTO 30
```

La Figure 9.4 illustre un exemple de ce programme. Pour vous perfectionner, vous devriez essayer d'obtenir des figures différentes telles que tracer des diagonales, ou encore se déplacer d'un point à un autre, sans tirer de trait.

Le deuxième exemple est basé sur le modèle, découvert au dix-neuvième siècle par le physicien français Lissajous, que l'on appelle "courbes de Lissajous".

```
10 LET t=0
20 LET t=t+0.1
30 LET x=50*(1+SIN(1.1*t))
40 LET y=50*(1+COS t)
50 PLOT x+50,y+50
60 GOTO 20
```

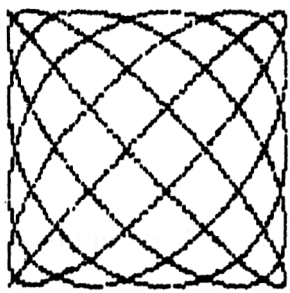


Figure 9.5 : Courbes de Lissajous

Le résultat que donne ce programme est montré en Figure 9.5. Vous pouvez modifier le modèle obtenu en changeant la valeur 1.1 en ligne 30.

Le dernier exemple de ce chapitre est un programme qui trace point par point le graphe de SIN(x)/x. Ceci donne la forme particulièrement intéressante que vous pouvez voir en Figure 9.6. Il ne faut pas oublier que la fonction SIN(x)/x est impossible à afficher quand x est égal à zéro.

```
10 FOR i=i TO 255
20 IF i=127 THEN GOTO 60
30 LET x=(i-127)/5
40 LET y=150*SIN(x)/x
50 PLOT i,y
60 NEXT i
```

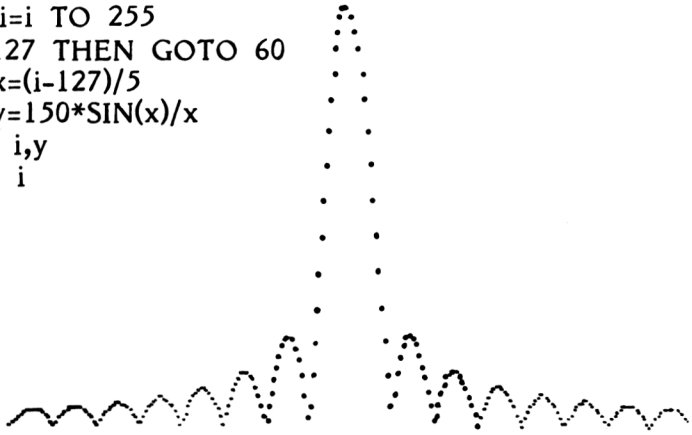


Figure 9.6 : Programme de $\frac{\text{SIN } x}{x}$

La meilleure stratégie à adopter lorsque vous utilisez les graphiques est de vous limiter à n'employer la haute résolution que dans les cas indispensables. En d'autres termes, privilégiez toujours la basse résolution, et servez-vous des graphiques haute résolution uniquement si cela peut conduire à de meilleurs résultats.

10
LOGIQUE ET AUTRES SUJETS

A présent, à l'aide du BASIC et de toutes les autres informations traitées dans les chapitres précédents, vous devriez être en mesure d'écrire n'importe quel programme. Cependant, le Spectrum offre encore d'autres facilités qui, bien que n'étant pas absolument indispensables, simplifient les traitements ou bien vont même au-delà des possibilités du BASIC. Ce chapitre rassemble tous ces compléments et explique brièvement comment ils fonctionnent.

Le premier sujet à traiter porte le nom plutôt inquiétant de logique, et présente les commandes AND (et), OR (ou) et NOT (non). La seconde partie introduit PEEK et POKE, IN, OUT, USR et CLEAR, qui affectent directement le fonctionnement du Spectrum. Nous considérons la commande ATTR qui sert à découvrir la couleur de n'importe quel point de l'écran en troisième partie ; et pour finir, nous verrons l'un des ordres les plus simples de BASIC, REM (REMarque). Et surtout,

ne croyez pas que cet ensemble de sujets n'a pas d'importance parce qu'il est traité en dernier, vous commetriez une erreur !

LOGIQUE ET EXPRESSIONS CONDITIONNELLES

Dans nos conversations de tous les jours, nous disons souvent des choses telles que "Avez-vous acheté des pommes et des oranges ?", "Préférez-vous le thé ou le café ?". L'usage des mots, et, ou, est tellement naturel que personne ne s'y arrête. Ce serait évidemment un grand avantage que de pouvoir utiliser ces mots dans des expressions conditionnelles BASIC. Par chance, la plupart des versions BASIC offrent cette possibilité, et sur le Spectrum cela se traduit par des expressions telles que :

```
a<0 AND b=3
a<0 OR b=3
```

La signification de chacune de ces expressions correspond exactement à celle du langage usuel (AND, et - OR, ou). La première expression ci-dessus est vraie si les deux conditions "a<0" et "b=3" sont vraies et la seconde expression est vraie si l'une ou l'autre des deux conditions est vraie. De même que AND et OR le Spectrum autorise aussi l'utilisation de NOT (non) qui change simplement la valeur d'une expression conditionnelle vraie en fausse et vice versa. Par exemple, "3=2" est faux mais "NOT 3=2" est vrai.

Vous pouvez combiner AND, OR et NOT avec n'importe laquelle des conditions que nous avons rencontrées au Chapitre 4 pour composer des expressions plus compliquées pouvant prendre l'une des valeurs vrai ou faux. (Comme vous le savez déjà par le Chapitre 4, vrai est représenté par 1 et faux par 0 dans le BASIC du Spectrum ce qui vous permet de les utiliser très facilement dans des expressions arithmétiques.) Les expressions conditionnelles qui incluent AND, OR ou NOT sont appelées habituellement expressions

logiques et nous pouvons maintenant réécrire la définition de la relation IF sous la forme :

IF <expression logique> THEN <instruction BASIC>

Pour contrôler qu'une coordonnée x n'est pas en dehors de l'écran, vous pourriez écrire, par exemple :

```
IF x<0 OR x>255 THEN ...
```

à la place de deux conditions IF qui seraient requises sans l'utilisation de OR.

Former des expressions logiques pour tester l'ensemble des conditions est habituellement la méthode la plus directe. Cependant il y a quelques pièges dans lesquels tombent même les plus expérimentés. Si vous voulez traduire la relation française "a égale b et c" alors vous devez répéter la condition "=". En d'autres termes vous devez écrire :

```
a=b AND a=c
```

et non

```
a=b AND c
```

qui donnera un résultat dépendant de c, suivant qu'il est à 0 ou 1. Vous devez aussi faire attention en utilisant NOT. Par exemple :

```
NOT(a=b AND a=c)
```

n'est pas identique à :

```
NOT(a=b) AND NOT(a=c)
```

Pour vérifier cela, essayez de travailler ces deux expressions pour quelques valeurs de a, b et c. La leçon est que vous devrez toujours faire attention à utiliser des

expressions logiques en vous détachant de leur signification dans le langage usuel.

Pour clore le sujet des expressions logiques, il est bon d'introduire la notion de table de vérité. Si vous considérez l'expression logique :

```
a AND b
```

où a et b sont des variables qui peuvent prendre les valeurs 0 pour faux, ou 1 pour vrai, vous pouvez dessiner une table qui recense toutes les valeurs possibles de a et b comme suit :

a	b	a AND b
0	0	0
0	1	0
1	0	0
1	1	1

Une telle table est appelée une table de vérité parce qu'elle indique les conditions pour lesquelles l'expression logique est vraie ou fausse. Vous pouvez dessiner des tables de vérité pour n'importe quelle expression logique et ceci est un moyen de vérifier que vous avez compris ce qui se passe. Par exemple OR et NOT ont les tables de vérité suivantes :

a	b	a OR b	NOT a
0	0	0	1
0	1	1	1
1	0	1	0
1	1	1	0

Le OR que nous avons utilisé précédemment n'est pas entièrement équivalent au mot français ou. La plupart des utilisations de "ou" en français signifient "l'un ou l'autre mais pas les deux". Par exemple, "Vous pouvez prendre de la confiture ou de la compote" signifie que vous pouvez en

prendre une mais (en principe) pas les deux ! Cependant OR logique signifie que vous pouvez avoir soit l'un, soit l'autre, soit les deux (regardez la table de vérité si vous n'en êtes pas sûr). OR logique est appelé plus précisément le OR inclusif parce qu'il inclut la possibilité des deux choix. Le ou du langage usuel est connu comme un ou exclusif parce qu'il exclut la possibilité des deux. Vous pouvez composer une expression logique équivalente au ou exclusif de la façon suivante :

or exclusif = (NOT(a) AND b) OR (a AND NOT(b))

comme on peut le vérifier dans la table de vérité :

a	b	(NOT(a) AND b) OR (a AND NOT (b))
0	0	0
0	1	1
1	0	1
1	1	0

Ce modèle de 0 (faux) et 1 (vrai) vous rappelle peut-être le modèle encre et papier du Chapitre 7 où la commande OVER a été traitée. En fait, après OVER 1 le point produit sur l'écran est un or exclusif entre l'ancien point et le point que vous avez essayé de placer.

A L'INTERIEUR DU SPECTRUM - BIN, PEEK, POKE, IN, OUT, USR et CLEAR

Cela peut paraître surprenant d'apprendre que le BASIC inclut des commandes qui permettent d'accéder au coeur même du Spectrum, compte tenu du fait que tout le BASIC examiné jusqu'à présent était fait de telle sorte qu'il évitait de faire référence à la façon dont la machine exécute les commandes. Cependant on rencontre quelques cas où les instructions courantes du BASIC sont insuffisantes. Par exemple, elles peuvent être trop lentes ou ne pas tenir

compte d'une caractéristique importante de la machine. Pour permettre au programme de trouver une parade à de telles difficultés, la plupart des versions du BASIC incluent des instructions qui donnent des possibilités d'accès à l'intérieur de la machine.

Nous avons déjà rencontré la fonction BIN (binaire) dans le Chapitre 7 où elle était utilisée, sans explication, dans la construction des caractères définis par l'utilisateur. Bien que, du point de vue du programmeur BASIC, le Spectrum semble traiter son arithmétique en nombres décimaux, en fait il travaille dans un système plus fondamental appelé binaire. Les hommes comptent en décimal tout simplement parce qu'ils ont dix doigts. Si nous n'avions que deux doigts, nous compterions alors, de la même façon qu'un ordinateur, en binaire. Bien qu'il ne soit pas indispensable à quiconque d'en connaître beaucoup sur le binaire, puisque le BASIC assure la conversion de décimal en binaire et réciproquement, cela est tout de même important si vous voulez utiliser votre Spectrum directement sans l'aide du BASIC. Encore que dans ce cas, tout ce que vous avez réellement besoin de savoir est que la fonction BIN (binaire) prendra un nombre binaire et le convertira en décimal (rappel : un nombre binaire est un nombre qui contient uniquement des zéros et des uns). Par exemple essayez :

```
PRINT BIN <nombre>
```

lequel imprimera l'équivalent décimal de tout nombre binaire que vous avez entré. Vous trouverez que 11 donne 3, 111 donne 7, et 11111111 (huit uns) donne 255. (Notez que BIN au contraire de toutes les autres fonctions du Spectrum ne donne pas la possibilité d'utiliser une expression.)

Il y a déjà quelque temps, nous avons appris qu'une variable est une zone de mémoire à laquelle on se réfère par son nom. Cependant, il est quelquefois nécessaire d'outrepasser cette méthode de référence à la mémoire, et d'utiliser de préférence l'accès direct à l'aide des

commandes PEEK et POKE. PEEK est une fonction qui donne le contenu d'une adresse mémoire, et POKE une commande qui permet de modifier ce contenu. C'est aussi simple que cela, à ceci près que vous devez préciser d'une part l'adresse mémoire concernée, et d'autre part le type de donnée que l'on peut y trouver. Le premier point peut être facilement résolu car le Spectrum comme tous les ordinateurs a ses zones mémoire numérotées séquentiellement à partir de zéro. De la sorte, PEEK(543) donnera le contenu de la 543ème adresse. Le second problème est aussi facilement résolu, sachant que l'on peut stocker un nombre binaire à une adresse mémoire. Ce nombre admettant au plus huit "1", le plus grand nombre que l'on puisse stocker dans une zone mémoire est donc 255, qui est l'équivalent décimal de 11111111. Ainsi, POKE 1000,200 mémorisera le nombre 200 à l'adresse 1000. Mais POKE 1000,600 donnera un message d'erreur car 600 est plus grand que 255. En général, pour vous servir de PEEK et POKE, vous devez savoir où se situe la zone qui vous intéresse et ce qu'elle contient, ce qui n'est pas toujours évident. Pour cette raison, il vaut mieux éviter d'utiliser PEEK et POKE, à moins que vous ne soyiez sûr de ce que vous faites.

Malgré cela, il y a tout de même une ou deux applications de ces deux commandes qu'il est bon de connaître, et qui démontrent la façon typique de les utiliser. Le Spectrum dispose d'une horloge qui bat en cinquantièmes de seconde. Du point de vue programmation, cela correspond à utiliser trois adresses mémoire, dont les valeurs changent de façon régulière. Les trois zones mémoire fonctionnent ensemble, afin de pouvoir étendre la marge de temps que l'on pourrait compter dans une seule zone. Celle située à l'adresse 23672 compte les cinquantièmes de seconde et se remet à zéro. Vous pouvez assimiler cela à une montre dont les aiguilles repasseraient par zéro tous les 256 battements. La deuxième adresse 23673 compte le nombre de tours que la première a effectués, ce qui revient à dire qu'elle compte en unités de 256 cinquantièmes de secondes. De la même manière, la troisième adresse 23674 compte le nombre de tours de la

deuxième, soit 256*256 cinquantièmes de seconde. Ainsi vous pourriez, à l'aide de la commande PEEK, découvrir le contenu de chacune de ces adresses, puis le convertir en secondes. Par exemple :

```
(65536* PEEK 23674 + 256*PEEK
 23673 + PEEK 23672)/50
```

vous donne le temps écoulé en secondes, depuis le démarrage de l'horloge, c'est-à-dire en principe depuis la mise sous tension. De la même manière, vous pourriez utiliser POKE pour initialiser les trois zones aux valeurs de votre choix ; pour remettre l'horloge à zéro par exemple :

```
POKE 23674,0:POKE 23673,0:POKE
23672,0
```

La plupart des applications de PEEK et POKE se ressemblent en ce sens qu'elles nécessitent la connaissance de l'endroit où se trouve une partie de l'information qui vous est dissimulée. Par exemple, sachant que la zone 23692 est utilisée pour mémoriser le nombre de lignes qui ont défilé avant que le message habituel "Scroll ? y/n" n'apparaisse, vous pouvez faire : POKE 23692,255 pour vous assurer que le message n'apparaîtra pas 255 fois.

De la même façon que PEEK et POKE examinent et modifient le contenu de la mémoire, IN et OUT permettent de communiquer avec les unités externes du Spectrum, telles que l'imprimante ZX. Toutefois, sauf cas très particulier, il est rare d'avoir besoin de ces instructions. Un exemple de IN et OUT est présenté au Chapitre 8, dans le sous-programme qui sert à produire un cliquetis. Le Spectrum considère son haut-parleur comme une unité externe, et c'est la raison pour laquelle ces commandes sont utilisées pour obtenir les sons.

Nous avons déjà rencontré la fonction USR lorsque nous avons parlé des caractères définis par l'utilisateur. Cependant, c'était un cas particulier de cette fonction, car

en général elle sert à transférer le contrôle du BASIC vers un programme en code machine stocké quelque part dans le Spectrum. De part sa spécificité, vous ne rencontrerez pas souvent cette instruction.

EXTRAIRE UNE COULEUR - ATTR

De la même manière que les fonctions SCREEN$ et POINT servent à donner des renseignements sur l'écran, la fonction ATTR permet de trouver les couleurs d'encre et de papier ou encore si un caractère est surbrillant ou clignotant. La seule difficulté est que ATTR donnant en retour un nombre, celui-ci doit être décodé pour en connaître la signification. Il est possible de dresser une liste d'instructions qui permettent de décoder l'information :

```
INT(ATTR(ligne,col)/128)
```

donne 0 si la position est stable, et 1 si elle est clignotante.

```
INT((ATTR(ligne,col)-INT(ATTR(ligne,col)
/128)*128)/64)
```

donne 0 si la position est surbrillante et 1 dans le cas contraire. De la même façon :

```
INT((ATTR(ligne,col)-INT(ATTR(ligne,col)
:64)*64)/8)
```

donne le code de couleur du papier et

```
ATTR(ligne,col)-INT(ATTR)(ligne,col)/8)*8
```

donne le code de la couleur de l'encre du point spécifié.

REM (REMarque)

L'instruction BASIC REM est la plus simple de toutes en ce sens qu'elle ne fait absolument rien ! En effet, elle vous permet simplement d'inclure des commentaires dans les programmes sans que ceux-ci soient assimilés à des instructions. Si vous souhaitez, par exemple, que tous vos programmes commencent par une ligne portant le nom du programme, vous pouvez utiliser :

```
10 REM Nom du programme
```

dans laquelle l'instruction REM indique au Spectrum que la suite ne doit pas être considérée comme une ligne de BASIC, mais comme un commentaire valable seulement pour le programmeur.

Cela vous étonne peut-être de retrouver une commande aussi simple à la fin d'un livre de BASIC. La raison en est qu'elle permet de clarifier les programmes. Après avoir vaincu la difficulté initiale que représente l'écriture de programmes BASIC, vous devez vous attacher à les écrire de mieux en mieux. Au début, un programme qui fonctionne est une récompense en soi mais avec l'expérience vous aimerez encore plus un programme bien écrit. De même que l'on apprend à programmer à force d'essais, d'erreurs et en lisant ce que d'autres ont réalisé, vous découvrirez par vous-même ce que l'on appelle un programme bien écrit.

Bien que l'instruction REM ne soit pas indispensable, elle vous aide à faciliter la compréhension de vos programmes. Une bonne idée consiste à inclure un commentaire dans chaque section de vos programmes indiquant ce qui s'y passe. De bonnes explications permettront de comprendre plus rapidement ce que vous avez voulu faire dans vos propres programmes, pour vous-même si vous les relisez quelque temps après les avoir écrits ou pour d'autres personnes à qui vous pourriez les donner.

Conclusion

Comme je l'ai dit plusieurs fois dans ce livre, la seule démarche pour apprendre à programmer est d'écrire des programmes. Bien sûr, les livres peuvent vous aider, mais uniquement si vous êtes prêt à faire vos propres expériences. Ne vous désolez pas si les premiers programmes manquent d'ambition, il est préférable au début de vous entraîner sur de petits exemples. Si vous essayez d'emblée des choses trop compliquées, vous aurez toutes les chances de commettre des erreurs difficiles à localiser. Essayez de découper vos programmes en sections n'exécutant que peu de choses à la fois ; en vous reportant à ce livre, vous trouverez beaucoup d'exemples de cette façon de faire. Une fois que tous vos petits morceaux de programmes seront au point, vous pourrez alors les rassembler en un seul gros programme. En tout état de cause, retenez surtout deux choses très importantes :

1. Il faut toujours aller de l'avant.

2. Prenez le maximum de plaisir à utiliser votre Spectrum.

INDEX

ABS, 108
ACS, 111
AND, 186
Apostrophe, 126-127
Arc cosinus, 111
  sinus, 111
  tangente, 111
ASN, 111
AT, 128-129
ATTN, 111
ATTR, 193

Backspace, 21
BEEP, 153
BIN, 140,189
Binaire, 8,190
Bloc de puissance, 15
BORDER, 146
Boucle conditionnelle, 69
  infinie, 59-60
  jusqu'à, 70
  tant que, 70
Boucles, 58-71
BREAK, 22
BRIGHT, 145

Caps Lock, 24
  Shift, 17,22
Caractères graphiques, 135-139
  utilisateur, 139
Chaîne constante, 84
  nulle, 89,117
  variable, 84
Chaînes, 83-102
CHR$, 112
CIRCLE, 171-174
Clavier, 16
CLEAR, 189
CLS, 131,150
CODE, 112
Code machine, 193
Concaténation de chaîne, 85
Conditions, 62-63
Coordonnées x,y, 129,170
Courbes de Lissajous, 182
Cos, 111
Curseur, 18

DATA, 98,160
Découpage de chaîne, 87
DEF FN, 117
Degrés, 111
DELETE, 31
DIM, 94

DRAW, 171-175

Edit, 31
Effets sonores, 161-162
ENTER, 18

EXP, 109
Expression arithmétique, 41,63
  conditionnelle, 64,186
  de chaîne, 85
  logique, 186
Extraction de sous-chaîne, 86

FLASH, 145
Flux de contrôle, 57-81
FN, 117
Fonctions, 106-120
  arithmétiques, 108-110
  définies par utilisateur, 117
  de chaînes, 112-113
  d'affichage, 128-131
  spéciales, 113-128
  trigonométriques, 110-112
FOR, 71-76

Générateur de nombres aléatoires, 114
GOSUB, 121
GOTO, 58-61
Graphiques, 125-151,169-184
  basse résolution, 125-151
  haute résolution, 169-184

IF, 62-71,187
Imprimante, 10
IN, 189
INK, 145
INKEY$, 113,116-117
INPUT, 44-47,133
INT, 109
INV VIDEO, 24
INVERSE, 142,179
I-O, 6
Itération, 60

LEN, 92,113
LET, 20,37-38
LIST, 22,30
LN, 109
LOAD, 101,128
Logarithmes, 109
Logique, 185-189

Magnétophone, 9,26
Mémoire, 7
Messages d'erreur, 28,88, 99,109,129,171

Microdrive, 9
Mode différé, 22
  étendu, 22
  graphique, 24
  immédiat, 22,118,143
Mot clé, 18-19

NEW, 29
NEXT, 72
Nombre exponentiel, 109
NOT, 186
Numéro de ligne, 19

Octet, 8
Opérateurs arithmétiques, 42
OR, 186
Ordre d'évaluation, 41
OU exclusif, 144,189
OUT, 189
OVER, 142,171

PAPER, 145

Paramètres, 107
  virtuels, 119
PAUSE, 160
PEEK, 189
Pi, 110
PLOT, 171
POINT, 180
Point-virgule, 126
POKE, 140,189
PRINT, 20,38-39,48
Priorités, 41-42,65
Pseudo-couleur, 149
  -hasard, 114

Radians, 111
RANDOMISE, 115
RAM, 7
READ, 99
Relation, 63
REM, 194
RESTORE, 98
RETURN, 121
RND, 113
RUN, 21

Sauvegarde, 100
SAVE, 101-127
SCREEN$, 163
Scroll, 22,59,131-132
SGN, 110
Sin, 111
Sous-chaîne, 86
Sous-programme, 121-123

SQR, 110
STEP, 74
STOP, 78,116,132
STR$, 113
Substitution de sous-chaîne, 87
Symbol Shift, 18

TAB, 128
Table de vérité, 188
Tableaux, 93
TAN, 111
THEN, 62,76
Touches de contrôle curseur, 21-25
  spéciales, 17
TRUE VIDEO, 24

Unité centrale, 7
  d'entrée, 6
  de sortie, 6
USR, 140,189

VAL, 113
Variables, 36-37
  numériques, 36
  simples, 36
VERIFY, 101-127
Virgule, 126

Zone d'affichage, 126-134
  d'entrée, 31
ZX80, 11
ZX81, 11
ZX-BASIC, 11

POUR UN CATALOGUE
DE NOS PUBLICATIONS

FRANCE
4, place Félix-Éboué
75583 Paris Cedex 12
Tél. : (1) 347.30.20
Telex : 211801

U.S.A.
2344 Sixth Street
Berkeley, CA 94710
Tel. : (415) 848.8233
Telex : 336311

ALLEMAGNE
Heyestr. 22
4000 Düsseldorf 12
Tel. : (0211) 287066
Telex : 08588163

SOULISSE et CASSEGRAIN, Imprimeurs, Niort (Deux-Sèvres).
Dépôt légal : Novembre 1983. N° 2133.

Cet ouvrage a été conçu pour permettre au lecteur n'ayant aucune connaissance en informatique d'apprendre rapidement à programmer en BASIC un micro–ordinateur ZX SPECTRUM. La méthode adoptée permet au lecteur de progresser pas à pas à l'aide de nombreux exemples de programmation, principalement dans le domaine des jeux. Trois chapitres sont consacrés au traitement des graphiques et des sons.

L'AUTEUR

S.M.GEE est co-auteur d'un manuel de programmation et collabore régulièrement à la revue Computing Today.

ISBN 2-902414-52-8
252 11-83 **88 F**

SYBEX

PROGRAMMEZ EN BASIC SUR SPECTRUM